不确定型逻辑程序的综合*

王立国

(北京航空航天大学计算机科学系)

SYNTHESIS OF NONDETERMINISTIC LOGIC PROGRAMS

WangLiguo

(*Beijing Institute of Aeronautics and Astronautics*)

ABSTRACT

A constructive proving system and the method of combining deduction with knowledge are presented for program synthesis. The synthesis of nondeterministic logic programs is discussed. An example of synthesis of Prolog program "append" is given to illustrate our method.

摘 要

本文提出了基于演绎和知识相结合的通过构造性证明综合程序的方法，进而讨论了不确定型逻辑程序的综合和自动综合的有关问题．用 append 程序的综合展示了这些方法．

§1. 引 言

程序设计的基本任务是由规范构造程序．程序综合是这一构造过程的严格化，形式化．程序自动综合是这一构造过程的自动化．

程序综合是程序验证的进一步发展．验证是给定规范和程序后证明程序满足规范；综

* 1988年5月收到．国家自然科学基金资助项目No.6863019和No.6875018，以及国家高技术智能机资助项目 No.863-306-0421-2.

合是只给定规范去构造出满足规范的程序．综合已经包含了验证，并且着重于更为关键的构造问题．

逻辑程序设计是人工智能和智能计算机的一项基础技术．不确定性是逻辑程序的一个重要特点，它使程序简洁，富于说明性．逻辑程序综合和自动综合则在深一层次为人工智能和智能计算机奠基．

演绎和知识是程序综合方法的两个重要方面．在[1]中作者提出了基于规范演绎的结构式程序综合方法．本文进一步探讨了演绎与知识相结合的方法并在此基础上讨论了不确定型逻辑程序的综合问题．

§2. 不确定型逻辑程序

逻辑程序具有描述性和过程性两个方面．不确定性是这两方面的统一．

例如，Prolog 程序 append(L_1, L_2, L):

append([ ], L_2, L_2): −.
append([H|T], L_2, [H|R]): −append(T, L_2, R).

可以描述性地解释成表 L_1 联接表 L_2 等于表 L；也可以过程性地解释成 L_1 联接 L_2 得到 L 和 L 分解成 L_1 和 L_2．

这样，有两种不确定性：1.入出不确定：L_1 和 L_2 为入，L 为出；L 为入，L_1 和 L_2 为出．2.输出的不确定性：Prolog 系统执行?−append([a,b], [c], L).得到 L＝[a, b, c]．但是执行?−append(L_1, L_2, [a, b, c]). 却得到 L_1＝[ ], L_2＝[a, b, c]; L_1＝[a], L_2＝[b, c]; L_1＝[a, b], L_2＝[c]; L_1＝[a, b, c], L_2＝[ ].

§3. 基于演绎和知识的程序综合系统

由规范形式化、机器化地构造出程序需要综合系统的支持．这一节中主要论述规范模式、规范演绎、知识库和综合结构．

1.规范模式

规范模式概括了程序规范的一般性结构．它表达成对偶〈$SPEC_f$，$PROG_f$〉．

$$\begin{cases} SPEC_f : \forall U(I(U) \Rightarrow \exists V(O(V)\&R(U,V))) \\ PROG_f : f(U;V) \end{cases}$$

其中 $SPEC_f$ 是程序 f 的规范：任何 U 满足输入条件 I(U)，则有 V，它满足输出条件 O(V)，并且 U 与 V 有关系 R(U, V)．f(U; V)是待求程序，U 为入，V 为出，用";"分

开。

为便于应用，常采用另一形式：

$$\begin{bmatrix} SPEC_f : input : U : I(U) \\ \qquad\qquad output : V : O(V) \\ \qquad\qquad relation : R(U,V) \\ PROG_f : f(U;V) \end{bmatrix}$$

例如，append$_1$(L_1, L_2, L)的规范是：

$$\begin{bmatrix} SPEC_{append_1} : input : L_1, L_2 : list(L_1), list(L_2) \\ \qquad\qquad output : L : list(L) \\ \qquad\qquad relation : 1.member(X, L_1) \vee member(X, L_2) \Leftrightarrow member(X, L) \\ \qquad\qquad\qquad 2.order(X, Y, L_1) \vee order(X, Y, L_2) \vee \\ \qquad\qquad\qquad (member(X, L_1) \& member(Y, L_2) \Leftrightarrow order(X, Y, L) \\ PROG_{append_1} : append(L_1, L_2; L) \end{bmatrix}$$

(为方便，约定省去全称约束变元的前置。)

其中 list(L)意为 L 是一个表，member(X, L)意为 X 是表 L 中一元，order(X, Y, L)意为在表 L 中，元 X 排在元 Y 之前(左)。

2.规范演绎

程序综合是一个构造性证明过程，其基本环节是由规范逆向构造子规范。重复此环节可得到综合过程的规范树。最后程序即可由此树同构得到。规范演绎即是用于实现构造子规范。其中图式一 SCHEME$_1$ 用于构造后置子规范，图式二 SCHEME$_2$ 用于构造前置子规范。

$$\begin{bmatrix} SPEC-DERI-SCHEME_1 : \\ \begin{bmatrix} TYPE-MATCH_1 : \forall U_1(I_{P_1}(U_1) \Rightarrow I_a(U_1)) \\ SUBSPEC-DERI_1 : \\ \quad \begin{bmatrix} SPEC_p : input : U_1, U_2 : I_{p_1}(U_1), I_{p_2}(U_1, U_2) \\ \qquad\qquad output : W : O_p(W) \\ \qquad\qquad relation : R_p(U_1, U_2, W) \\ PROG_p : p(U_1, U_2; W) \end{bmatrix} \\ \quad \Uparrow \begin{bmatrix} SPEC_a : input : U_1 : I_a(U_1) \\ \qquad\qquad output : V : O_a(V) \\ \qquad\qquad relation : R_a(U_1, V) \\ PROG_a : a(U_1; V) \end{bmatrix} \\ \quad \begin{bmatrix} SPEC_c : input : V, U_2 : O_a(V), I_c(V, U_2) \\ \qquad\qquad output : W : O_p(W) \\ \qquad\qquad relation : R_c(V, U_2, W) \\ PROG_c : c(V, U_2; W) \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$THEOREM-PROV_1$:
1. $\forall U_1, U_2, V(I_{P_2}(U_1, U_2)\ \&\ R_a(U_1, V) \Rightarrow I_c(V, U_2))$
2. $\forall U_1, U_2, V, W(R_a(U_1, V)\ \&\ R_c(V, U_2, W) \Rightarrow R_p(U_1, U_2, W))$

$PROGRAM_1: p(U_1, U_2; W) :- a(U_1; V), c(V, U_2; W).$

$SPEC-DERI-SCHEME_2$:

$TYPE-MATCH_2: \forall W_1(O_a(W_1) \Rightarrow O_{P_1}(W_1))$

$SUBSPEC-DERI_2$:

$SPEC_p$: *input*: $U: I_p(U)$
output: $W_1, W_2: O_{p_1}(W_1), O_{p_2}(W_1, W_2)$
relation: $R_p(U, W_1, W_2)$

$PROG_p: p(U; W_1, W_2)$

$SPEC_a$: *input*: $V: I_a(V)$
output: $W_1: O_a(W_1)$
relation: $R_a(V, W_1)$

$PROG_a: a(V; W_1)$

$SPEC_c$: *input*: $U: I_p(U)$
output: $V, W_2: I_a(V), O_c(V, W_2)$
relation: $R_c(U, V, W_2)$

$PROG_c: c(U; V, W_2)$

$THEOREM-PROV_2$:
1. $\forall V, W_1, W_2(O_c(V, W_2)\ \&\ R_a(V, W_1) \Rightarrow O_{P_2}(W_1, W_2))$
2. $\forall U, V, W_1, W_2(R_c(U, V, W_2)\ \&\ R_a(V, W_1) \Rightarrow R_p(U, W_1, W_2))$

$PROGRAM_2: p(U; W_1, W_2) :- c(U; V, W_2), a(V; W_1).$

SPEC-DERI-SCHEME$_1$ 表明，为了证明〈SPEC$_p$, PROG$_p$〉在输入域匹配 TYPE-MATCH1 条件下，引入已知的〈SPEC$_c$, PROG$_c$〉，得到一个新的待证后置子规范〈SPEC$_c$, PROG$_c$〉。这一子规范推导需要并引出了定理证明 THEOREM-PROV$_1$。与此过程同构地得到程序 PROGRAM$_1$

SPEC-DERI-SCHEME$_1$ 相当于下面的证明：

{TYPE-MATCH$_1$, THEOREM-PROV$_1$, SPEC$_a$, SPEC$_c$} ⊢ SPEC$_p$

给定〈SPEC$_p$, PROG$_p$〉，〈SPEC$_a$, PROG$_a$〉可以将 THEOREM-PROV$_1$ 强化为：

1. $\forall U_1, U_2, V(I_{p_2}(U_1, U_2) \& R_a(U_1, V) \Leftrightarrow I_c(V, U_2)$
2. $\forall U_1, U_2, V, W(R_c(V, U_2, W) \Leftrightarrow (R_a(U_1, V) \Rightarrow R_p(U_1, U_2, W))$

这时得到最弱后置子规范。

SPEC-DERI-SCHEME$_1$ 给出了一个层次化的构造证明：高层——由类型匹配 TYPE-MATCH$_1$ 导引的子规范推导 SUBSPEC-DERI$_1$；低层——由 SUBSPEC-DERI$_1$ 导引的定理证明 THEOREM-PROV$_1$。

这些导引使综合和证明较易于机器化．这主要是：

(1) 由 R_p 构造 R_c，前提主要限于 R_a，由 I_{p_2} 构造 I_c 亦然．

(2) 合一匹配被指定：$R_a(U_1, V)$与 $R_p(U_1, U_2, W)$中含 U_1 的谓词需匹配、消去，并且仅需如此即可得到 $R_c(V, U_2, W)$．类似地，可得到 $I_c(V, U_2)$．

(3) U_1, U_2, V 和 W 均是全称变元，对它们不需要复杂的 Skolem 函数．

这些提高了机器化的时空效率，为自动综合奠定了基础．

对 SPEC–DERI–SCHEME$_2$ 可给出类似讨论，主要区别是它得到前置和最弱前置子规范．

3.知识库

知识库是为配合规范演绎而设置．规范演绎是一般形式的，而知识库则针对具体领域．本文只讨论逻辑程序设计方面的知识．

(1)目标语言

1.
$$\begin{cases} SPEC_{cons} : input : H, T : list(T) \\ \qquad\qquad output : L : list(L) \\ \qquad\qquad relation : 1.(X = H) \vee member(X, T) \Leftrightarrow member(X, L), \\ \qquad\qquad\qquad 2.order(X, Y, T) \vee \{(X = H)\ \&\ member(Y, T)\} \Leftrightarrow \\ \qquad\qquad\qquad order(X, Y, L) \\ PROG_{cons} : cons(H, T; L) \end{cases}$$

2.
$$\begin{cases} SPEC_{car-cdr} : \quad input : L : list(L), L \neq [\ ] \\ \qquad\qquad output : H, T : list(T) \\ \qquad\qquad relation : \ 1.member(X, L) \Leftrightarrow (X = H) \vee member(X, T) \\ \qquad\qquad\qquad 2.order(X, Y, L) \Leftrightarrow ((X = H) \& member(Y, T)) \vee \\ \qquad\qquad\qquad order(X, Y, T) \\ \quad PROG_{car-cdr} : car - cdr(L; H, T) \end{cases}$$

3.
$$\begin{cases} SPEC_{is} : \ \forall X, Y(P(X) \Rightarrow P(Y)) \\ PROG_{is} : \ Y\ is\ X \end{cases}$$

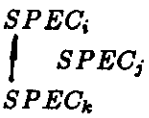
4.
$$\begin{cases} SPEC_{ture} : \ true \\ PROG_{ture} : \ stop \end{cases}$$

(2)数据类型

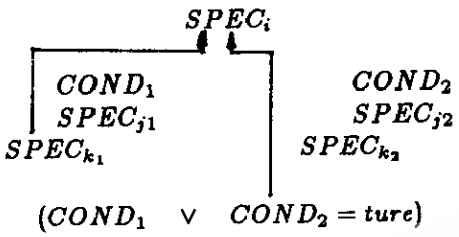
$$1.TYPE_{cons} : (H : element * T : list \rightarrow L : list)$$
$$2.TYPE_{car-cdr} : (L : list \rightarrow H : element * T : list)$$
$$3.TYPE_{is} : (TYPE(Y) = TYPE(X))$$

(3)程序设计

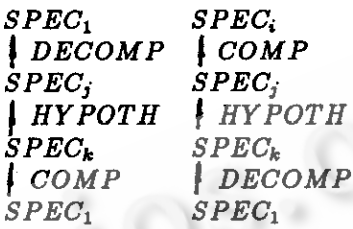
(a)复合



(b)分支

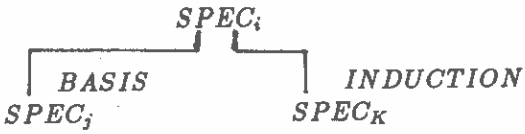


(c)逆归



DECOMP 是关于程序输入变元的分解，HYPOTH 是对 $SPEC_i$ 的归纳假设，COMP 是关于输出变元的复合．为保证程序的可终止性，要求 DECOMP 是良基分解，并且，$SPEC_j$ 与 HYPOTH 中良基序变元在规范演绎中视为固定常元．

(4)良基序



$BASIS : L[\ ],\ L[H]$

$INDUCTION : L \rightarrow car - cdr(L; H, T) \rightarrow T$

(5)表

$$1.THEOREM_{nil} : \forall L(list(L)\ \&\ L = [\ ] \Rightarrow \forall X(member(X, L) \Leftrightarrow false)\ \& \\ \forall X, Y(order(X, Y, L) \Leftrightarrow false))$$

$$2.THEOREM_{unit} : \forall L(list(L)\ \&\ L = [H] \Rightarrow \forall X(member(X, L) \Leftrightarrow (X = H)\& \\ \forall X, Y(order(X, Y, L) \Leftrightarrow false))$$

下面略做解释．

类似 Lisp, cons(H, T; L)和 car–cdr(L; H, T)是 Prolog 的基本操作，它们在实用系统中常统一成[H|T]．

〈$SPEC_{is}$, $PROG_{is}$〉是赋值公理，其中 P 是任意谓词．

SPECtrue 是永真命题，对应程序 stop．

数据类型的知识用于导引类型匹配．

程序设计的知识是关于程序的基本控制结构：复合、分支和递归．

良基序的知识给出递归结构的表的分解：表 L 分解成头 H 和尾 T，由 L 到 T 是良基序，最小元是空表[ ]和单元表[H]．

表的知识是关于空表和单元表的公理．

4.综合的结构

(1)对称性

规范推导图式中建立了规范推导 SUBSPEC-DERI 和生成的程序 PROGRAM 之间的对称.

以此对称为基本环节可建立起规范树和程序树之间的对称.

$$\begin{array}{ll} SPEC_p & PROG_p \\ \mid\ SPEC_a & \mid\ PROG_a \\ SPEC_c & PROC_c \end{array}$$

(2)层次性

基于规范演绎的程序综合是由规范推导到定理证明这样一个层次化的结构式推导过程.

(3)演绎和知识的协调

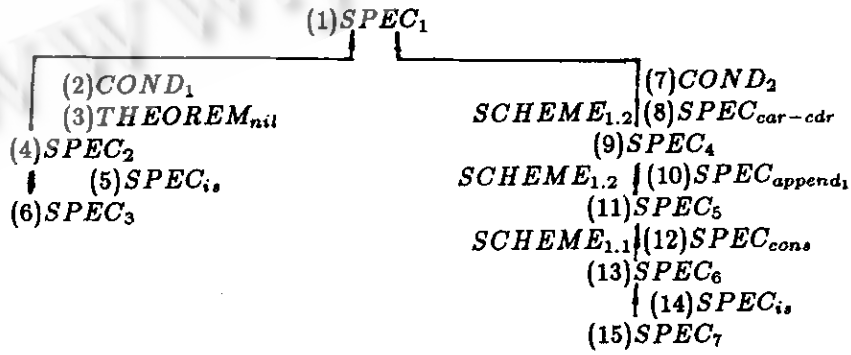
在综合过程中演绎和知识相互依存，转化. 演绎需要知识库中知识为前提，特别是需从公理式基本知识出发；知识库中不仅存有公理式的基础知识，还可以累积、增殖新的定理式知识. 演绎和知识的平衡是动态的. 一个具体的方案可以是：由公理式知识出发，经过严格的演绎，把产生的新知识——对偶累积于知识库中以用于后面的综合而不再重复其证明. 演绎和知识的协调可看成是程序综合领域中一种高层次的时间和空间的协调. 基于这种协调，我们可以在自动综合系统中建立起专用的机器学习机制.

§ 4. 不确定型逻辑程序的综合

基于构造性证明的规范演绎，不仅可以综合出逻辑程序，进而可以综合出不确定性的逻辑程序. 这里主要以 Prolog 程序 append(L_1, L_2, L)的综合为例来说明这些方法.

$append_1(L_1, L_2; L)$的规范如前面规范模式一节所述.

综合以 L_1 和 L_2 为入，L 为出的 $append_1(L_1, L_2; L)$程序的规范树如下:



规范树中诸项含义如下:

(1) $SPEC_1 = SPEC_{append_1}$

(2) $COND_1 : L_1 = [\ \ ]$

(3) $THEOREM_{nil}\ (L1/L)$

(4) $SPEC_2 : input : L_1, L_2 : L_1 = [\ \ ], list(L_2)$
$output : L : list(L)$
$relation : 1.member(X, L_2) \Leftrightarrow member(X, L)$
$2.order(X, Y, L_2) \Leftrightarrow order(X, Y, L)$
$PROG_2 : f_1([\ \ ], L_2; L)$

(5) $SPEC_{is}\ (L_2/X, L/Y)$

(6) $SPEC_3 : true$
$PROG_3 : stop$

(7) $COND_2 : L_1 \neq [\ \ ]$

(8) $SPEC_{car-cdr}\ (L_1/L)$

(9) $SPEC_4 :\ input : H, T, L_2 : list(T), list(L_2)$
$output : L : list(L)$
$relation : 1\ (X = H) \vee member(X, T) \vee member(X, L_2) \Leftrightarrow member(X, L)$
$2.order(X, Y, T) \vee ((X = H)\ \&\ member(Y, T)) \vee order(X, Y, L_2)$
$\vee (((X = H) \vee member(X, T))\ \&\ member(Y, L_2) \Leftrightarrow$
$order(X, Y, L)$
$PROG_4 : f_2(H, T, L_2; L)$

(10) $SPEC_{append_1}((T, L_2; M)/(L_1, L_2; L))$

(11) $SPEC_5 : input : H, M : list(M)$
$output : L : list(L)$
$relation : 1(X = H) \vee member(X, M) \Leftrightarrow member(X, L)$
$2\ order(X, Y, M) \vee ((X = H)\ \&\ member(Y, M) \Leftrightarrow order(X, Y, L)$
$PROG_5 : f_3(H, M; L)$

(12) $SPEC_{cons}\ ((H, M; R)/(H, T; L))$

(13) $SPEC_6 : input : R : list(R)$
$output : L : list(L)$
$relation : 1\ member(X, R) \Leftrightarrow member(X, L)$
$2\ order(X, Y, R) \Leftrightarrow order(X, Y, L)$
$PROG_6 : f_4(R; L)$

(14) $SPEC_{is}(R/X, L/Y)$

(15) $SPEC_7 : ture$
$PROG_7 : stop$

$SCHEME_{1.1}$ 和 $SCHEME_{1.2}$ 是 SPEC–DERI–SCHEME$_1$ 的特例.

$SCHEME_{1.1}$:

- $TYPE-MATCH_{1.1} : \forall U(I_p(U) \Rightarrow I_a(U))$
- $SUBSPEC-DERI_{1.1}$:
  - $SPEC_p$: *input* : U : $I_p(U)$
    output : W : $O_p(W)$
    relation : $R_p(U,W)$
  - $PROG_p : p(U;W)$
    - $SPEC_a$: *input* : U : $I_a(V)$
      output : V : $O_a(V)$
      relation : $R_a(U,V)$
    - $PROG_a : a(U;V)$
  - $SPEC_c$: *input* : V : $O_a(V)$
    output : W : $O_p(W)$
    relation : $R_c(V,W)$
  - $PROG_c : c(V;W)$
- $THEOREM-PROV_{1.1} : \forall U,V,W(R_a(U,V)\ \&\ R_c(V,W) \Rightarrow R_p(U,W))$
- $PROGRAM_{1.1} : p(U;W) : -a(U;V), c(V;W)$

$SCHEME_{1.2}$:

- $TYPE-MATCH_{1.2} : \forall U_1(I_{p_1}(U_1) \Rightarrow I_a(U_1))$
- $SUBSPEC-DERI_{1.2}$:
  - $SPEC_P$: *input* : $U_1, U_2; I_{p_1}(U_1), I_{p_2}(U_2)$
    output : W : $O_p(W)$
    relation : $R_p(U_1,U_2,W)$
  - $PROG_p : p(U_1,U_2;W)$
    - $SPEC_g$: *input* : $U_1; I_a(U_1)$
      output : V : $O_g(V)$
      relation : $R_a(U_1V)$
  - $PROG_a : (U_1;V)$
  - $SPEC_a$: *input* : $V, U_2; O_a(V), 1_{p_2}(U_2)$
    output : W : $O_p(W)$
    relation : $R_a(V,U_2,W)$
  - $PROG_a : c(V,U_2;W)$
- $THEOREM_{1.2} : \forall U_1,U_2,V,W(R_a(U_1,V)\ \&\ R_c(V,U_2,W) \Rightarrow R_p(U_1,U_2W))$
- $PROGRAM_{1.2} : p(U_1,U_2;W) : -a(U_1;V), c(V,U_2;W).$

下面进一步说明规范树中如何由规范推导出子规范.

$$(a)\ (1)SPEC_1 \leftarrow (4)SPEC_2:$$

由于 L_1 是表，可按良基序长度归纳．基始是 $COND_1$: $L_1=[\ ]$.

由 $THEOREM_{nil}$, member(X, L_1)和 order(X, Y, L_1)为假 false 被消去，得到(4) $SPEC_2$.

$$(b)\ (4)SPEC_2 \leftarrow (6)SPEC_3:$$

由 $SPEC_{is}$ 做代换(L_2 / X)，L / Y)，$SPEC_2$ 成为 true 得到 (6) $SPEC_3$.

$$(c)\ (1)SPEC_1 \leftarrow (9)SPEC_4:$$

由分支知识，引入 $COND_2$: L1≠[ ]．由于 $SPEC_1$ 的输入变元 L_1 与目标语言知识中 $SPEC_{car-cdr}$ 的输入变元同为非空表，这样 $SCHEME_{1.2}$ 中 $TYPE-MATCH_{1.2}$ 满足．故选用 $SPEC_{car-cdr}$ 和 $SCHEME_{1.2}$，产生如下合一匹配:

(1) 变元: (a) $SPEC_1$ / / $SPEC_p$: L_1 / U_1, L_2 / U_2, L / W

(b) $SPEC_{car-cdr}$ / / $SPEC_a$: L / U_1, (H, T) / V

(c) $SPEC_1$ / / $SPEC_{car-cdr}$: L_1 / L

(2) 谓词: $(a)SPEC_1 / / SPEC_p$: $list(L_1)/I_{p_1}(U_1)$, $list(L_2)/I_{p_2}(U_2)$, $list(L)/O_p(W)$

$(b)SPEC_{car-cdr} / / SPEC_a$: $(list(L)\ \&\ L\neq[\ ])/(I_a(U_1)$ $list(T)/O_a(V)$, $(SPEC_{car-cdr}-relation)/R_a(U_1, V)$

(3) 程序: (a) $PROG_1$ / / $PROG_p$: $append_1(L_1, L_2; L)$ / $p(U_1, U_2; W)$

(b) $PROG_{car-cdr}$ / / $PROG_a$: car-cdr(L; H, T) / $a(U_1; V)$

规范推导 $SUBSPEC-DERI_{1.2}$ 引出定理证明 $THEOREM-PROV_{1.2}$:

$\forall L_1, L_2, H, T, L\ (R_a(L_1, H, T)\ \&\ R_c(H, T, L_2, L)\Rightarrow R_p(L_1, L_2, L))$

由 $THEOREM-PROV_{1.2}$ 可知，只需在 R_p 与 R_a 之间匹配含 L_1 的谓词 member(X, L_1) 并消去，即可得到: $R_c(H, T, L_2, L)$=(SPEC$_4$-relation).

用合一作用于 $SCHEME_{1.2}$ 的<$SPEC_c$, $PROG_c$>，最后得到(9) $SPEC_4$.

(d) (9) $SPEC_4$←(11) $SPEC_5$:

与(c)类似，区别主要是(10)$SPEC_{append1}$ 是由 (1)$SPEC_1$ 按归纳假设得到(注意，(9)$SPEC_4$ 与(10)$SPEC_{append1}$ 中 T 视为固定常元)．由于 $SPEC_{car-cdr}$ 是关于表长的良基序分解并且归纳基始已完成，因此可以保证构造出的递归程序的终止性.

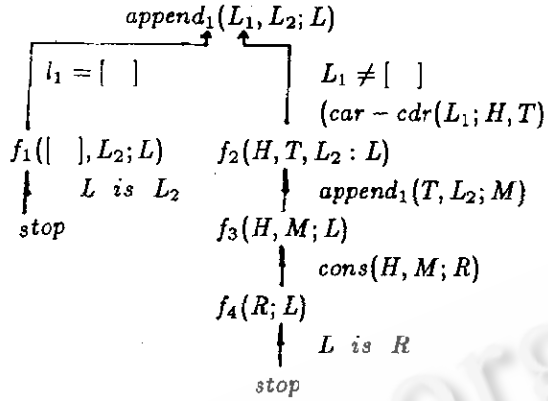
(e) (11) $SPEC_5$←(13)$SPEC_6$:

由知识库中 $SPEC_{cons}$ 可得.

(f) (13)$SPEC_5$←(15)$SPEC_7$:

由知识库中 $SPEC_{is}$ 可得.

由规范树可同构地得到程序树



由程序树得到 $append_1(L_1, L_2; L)$ 的 Prolog 程序:

$append_1(L_1, L_2; L) : -L_1 = [\ ], f_1([\ ], L_2; L).$
$f_1([\ ], L_2; L) : -L\ is\ L_2, stop.$
$append_1(L_1, L_2; L) : -L_1 \neq [\ ], car - cdr(L_1; H, T), f_2(H, t, L_2; L).$
$f_2(H, T, L_2; L) : -append_1(T, L_2; M), f_3(H, M; L).$
$f_3(H, M; L) : -cons(H, M; R), f_4(R; L).$
$f_4(R; L) : -L\ is\ R, stop.$

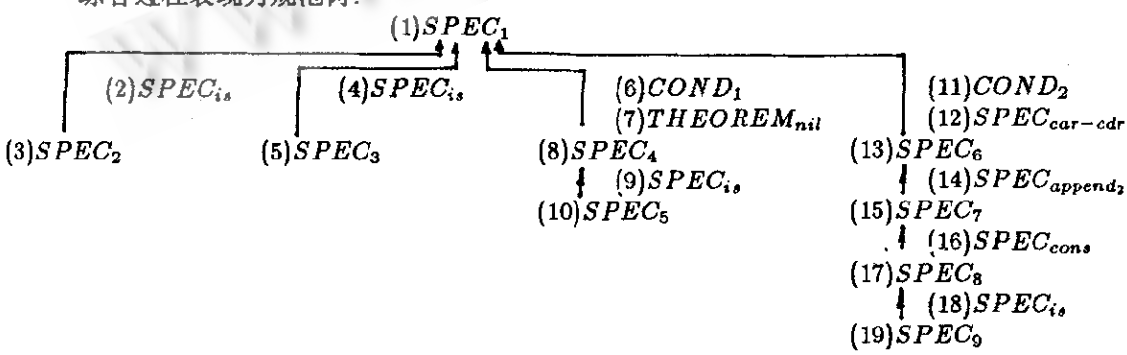
进一步化简，可得到:

$append_1([\ ], L_2; L_2) : -.$
$append_1([HIT], L_2; [HIM]) : -append_1(T, L_2; M).$

下面进一步综合与 $append_1(L_1, L_2; L)$ 入出相反的 $append_2(L; L_1, L_2)$. 由 $append_1(L_1, L_2; L)$ 的规范可对称地得到:

$SPEC_{append_2}$:input : $L : list(L)$
output : $L_1, L_2 : list(L_1), list(L_2)$
relation : 1$member(X, L) \Leftrightarrow member(X, L_1) \vee member(X, L_2)$
2$order(X, Y, L) \Leftrightarrow order(X, Y, L_1) \vee order(X, Y, L_2) \vee$
$(member(X, L_1)\ \&\ member(Y, L_1))$
$PROG_{append_2}$:$append_2(L; L_1, L_2)$

综合过程表现为规范树:



树中诸项含义如下：

(1) $SPEC_1 = SPEC_{append_2}$

(2) $SPEC_{i_s}([\ \ ]/X, L_1/Y; L/X, L_2/Y)$

(3) $SPEC_2 : true$

$PROG_2 : stop$

(4) $SPEC_{i_s}(L/X, L_1/Y; [\ \ ]/X, L_2/Y)$

(5) $SPEC_3 : true$

$PROG_3 : stop$

(6) $COND_1 : L = [\ \ ]$

(7) $THEOREM_{nil}(L/L)$

(8) $SPEC_4 : input : L : list(L), L = [\ \ ]$

$output : L_1, L_2 : list(L_1), list(L_2)$

$relation : 1 false \Leftrightarrow member(X, L_1) \vee member(X, L_2)$

$2 false \Leftrightarrow order(X, Y, L_1) \vee order(X, Y, L_2) \vee (member(X, L_1)\ \&\ member(Y, L_2))$

$PROG_4 : f_1([\ \ ]; L_1, L_2)$

(9) $SPEC_{i_s}([\ \ ]/X, L_1/Y; [\ \ ]/X, L_2/Y)$

(10) $SPEC_5 : true$

$PROG_5 : stop$

(11) $COND_2 : L \neq [\ \ ]$

(12) $SPEC_{car-cdr} : ((L; H, T)/(L; H, T))$

(13) $SPEC_6 : input : H, T : list(T)$

$output : L_1, L_2 : list(L_1), list(L_2)$

$relation : 1\ (X = H) \vee member(X, T) \Leftrightarrow member(X, L_1) \vee member(X, L_2)$

$2\ order(X, Y, T) \vee ((X = H)\ \&\ member(Y, T)) \Leftrightarrow order(X, Y, L_1) \vee order(X, Y, L_2) \vee (member(X, L_1)\ \&\ member(X, L_2))$

$PROG_6 : f_2(H, T; L_1, L_2)$

(14) $SPEC_{append_2}\ ((T; N_1, N_2)/(L; L_1, L_2))$

(15) $SPEC_7 : input : H, N_1, N_2 : list(N_1), list(N_2)$

$output : L_1, L_2 : list(L_1), list(L_2)$

$relation : 1\ (X = H) \vee member(X, N_1) \vee member(X, N_2) \Leftrightarrow member(X, L_1) \vee member(X, L_2)$

$2\ order(X, Y, N_1) \vee order(X, Y, N_2) \vee (member(X, N_1)\ \&\ member(Y, N_2) \vee ((X = H)\ \&\ (member(Y, N_1) \vee member(Y, N_2)) \Leftrightarrow order(X, Y, L_1) \vee order(X, Y, L_2) \vee (member(X, L_1)\ \&\ member(Y, L_2))$

$PROG_7 : f_3(H, N_1, N_2; L_1, L_2)$

(16) $SPEC_{cons}((H, N_1; R)/(H, T; L))$

(17) $SPEC_8 : input : R, N_2 : list(R), list(N_2)$

$output : L_1, L_2 : list(L_1), list(L_2)$

$relation : 1\ member(X, R) \vee member(X, N_2) \Leftrightarrow member(X, L_1) \vee member(X, L_2)$

$2\ order(X, Y, R) \vee order(X, Y, N_2) \vee (member(X, R)\ \&\ member(Y, N_2) \Leftrightarrow order(X, Y, L_1) \vee order(X, Y, L_2) \vee (member(X, L_1)\ \&\ member(Y, L_2))$

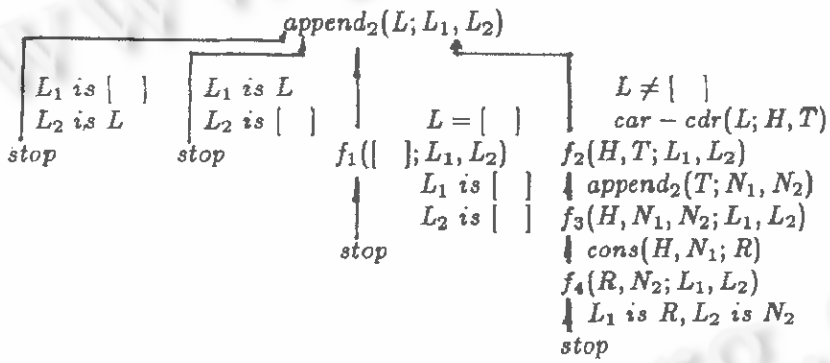
$PROG_8 : f_4(R, N_2; L_1, L_2)$

(18) $SPEC_{is}(R/X, L_1/Y; N_2/X, L_2/Y)$

(19) $SPEC_9 : true$

$PROG_9 : stop$

由规范推出子规范的过程与前面类似，不再赘述。由规范树可同构地得到程序树：



由程序树得到程序：

$append_2(L; L_1, L_2) : -L_1\ is\ [\ ], L_2\ is\ L, stop.$

$append_2(L; L_1, L_2) : -L_1\ is\ L, L_2\ is\ [\ ], stop.$

$append_2(L; L_1, L_2) : -L = [\ ], f_1([\ ]; L_1, L_2).$

$f_1([\ ]; L_1, L_2) : -L_1\ is\ [\ ], L_2\ is\ [\ ], stop.$

$append_2(L; L_1, L_2) : -L \neq [\ ], car - cdr(L : H, T), f_2(H, T; L_1, L_2).$

$f_2(H, T; L_1, L_2) : -append_2(T; N_1, N_2), f_3(H, N_1, N_2; L_1, L_2).$

$f_3(H, N_1, N_2; L_1, L_2) : -cons(H, N_1; R), f_4(R, N_2; L_1, L_2).$

$f_4(R, N_2; L_1, L_2) : -L_1\ is\ R, L_2\ is\ N_2, stop.$

化简后得到：

(a) $append_2(L;\ [\ ],\ L) : -.$

(b) $apend_2(L;\ L,\ [\ ]) : -.$

(c) $append_2([\ ];\ [\ ],\ [\ ]) : -.$

(d) $append_2([H|T];\ [H|N_1],\ N_2) : -append_2(T;\ N_1,\ N_2).$

可以证明：

(1) $(a) \vee (b) \Rightarrow (c)$

(2) $(a)\ \&\ (d) \Rightarrow (b)$

这样得到：

$append_2(L;\ [\ ],\ L)$: −.

$append_2(H|T];\ [H|N_1],\ N_2)$: $-append_2(T;\ N_1,\ N_2)$.

$append_1$ 与 $append_2$ 合并，最后得到不确定型逻辑程序：

$append([\ ],\ L_2,\ L_2)$: −.

append ([H|T], L_2; [H|R])−append (T, L_2, R).

append 程序有三个变元 L_1，L_2，L，不同的入出组合共有六种．本文只讨论了代表性的两种，其余可用类似方法推出．

从构造性证明角度来看，综合出不确定型的逻辑程序并没有遇到新的原则性困难．对于入出不确性而言，重要的是可以在不同的入出程序的综合过程中得到高层次的启发性控制信息．这一信息可用于选择知识，而这正是综合中较为关键和困难的高层决策．

例如，由综合 $append_1$ 的选择(12)$SPEC_{cons}$ 和(8)$SPEC_{car-cdr}$ 可决策在综合 $append_2$ 中相应选将(12) $SPEC_{car-cdr}$ 和(16) $SPEC_{cons}$．这些高层决策的优选将大大减轻低层推理的时空消耗．

一般地来讲，由于入出不确定性逻辑程序是以同一形式的程序来概括一个程序族．因此族中诸程序相应的规范树必然包括较多的类似结构，进一步，它们可以是由较低层次的入出不确定性逻辑程序的规范组成．逻辑程序的入出不确定性源于 Prolog 的基本函数[H|T]的入出不确定性：可以以 H，T 为入，[H|T]为出；也可相反．这反映在 $SPEC_{cons}$ 和 $SPEC_{car-cdr}$ 中二者的 input 和 output 是对称的，relation 是相同的．

逻辑程序的输出不确定性可在中综合系统中加入对规范的逻辑结构进行分析的子系统来完成，因为逻辑程序的输出不确定性源于规范中潜含的不确定性．例如 $SPEC^{append2}$ 的 relation 中，1 member(X，L)⇔member(x，L_1) V member (X，L_2)对同一输入 L，输出 L_1，L_2 可有不同选择 L_1=[ ]，L_2=L 和 L_1=L，L_2=[ ]．它们均满足 relation．

§5. 综合中的学习机制

综合系统可以以公理式基本知识为基础，经过严格演绎和知识库操作来自我增殖知识．以统一形式的规范模式为基本框架；以知识库中公理式知识为基础；以规范演绎为基本步骤，综合系统既可以在为用户工作时学到用户的知识(用户给出的规范要求)，又可以内省式地自行构造出新的规范和程序．

例如，用户给出要求：$SPECT_{sort}$ 系统完成任务的同时也学到了新知识〈$SPEC_{sort}$，$PROG_{sort}$〉．进而，在综合 $SPEC_{sort}$ 中，系统又自我发现了 $SPEC_{insert}$ 和 $SPEC_{select}$．在系统能力不断增强时具有了知识〈$SPEC_{append}$，$PROG_{append}$〉．它又构造出快速排序程序 quick−sort．

如此，基于演绎、知识和学习，程序综合系统可以表现出与环境交互又自我完善的智能增进的动态过程．

§6. 实验系统

初步的实验系统已经完成. 包括 append 和一些典型 Prolog 程序已经通过. 这是一个人机交互实验系统, 进一步完善工作正在进行.

§7. 结 论

程序设计是创造性智力活动. 程序综合和自动综合则是力图以形式化、进而机器化方法来改善这种活动.

这一任务是有意义的, 它将提高计算机自身的自动化水平, 在深入层次为人工智能研究奠基. 同时, 这一任务又是困难的, 我们只能是逐步逼近程序设计机器化这一目标.

需要对程序设计自身的规律和本质做更深入的研究. 同时, 需要引入有关知识的研究成果, 以程序综合为目标把各种方法、各个方面有机地组织起来.

这些研究在基础方面可能导致程序设计的元科学的问题, 在应用方面可能导致建立智能化的程序设计环境的问题. 这些任务是困难的, 诱人的, 需要付出不懈的努力.

参考文献

[1] 王立国. 基于规范演绎的结构式程序综合. 计算机学报, 1988 年第 6 期.

[2] 王立国. 自动程序设计. 计算机世界. 总第 152 期, 1987, 3, 8.

[3] Wang Li-Guo, Heuristic Backward and Forward Inference in Program Synthesis, 7th International Workshop, Expert Systems and Their Application, Avignon, France, May 1987.

[4] Sun Huai-Min and Wang Li-Guo, A Model Theory of Logic Programming Methodology, Proceedings of the Second International Logic Programming Conference, Uppsala, Sweden, July 1984.

《软件学报》征稿简则

一、《软件学报》是中国科学院软件研究所主办的学术性刊物，本刊对象主要是计算机科学研究人员、工程技术人员，大专院校教师，研究生及高年级学生。

二、本刊刊登计算机软件的研究、设计与实现，计算机理论，计算机网，数据库，人工智能，计算机辅助设计及软件新技术的应用开发等方面的研究论文、短文、技术报告、研究简报、通讯及综述。

三、来稿要求和注意事项

1.来稿内容力求正确、文字精炼，各类文稿具体要求如下：

学术论文：具有重大创新，有新的观点、见解或具有中国的特点，推动或丰富了该课题领域的研究与发展。字数不超过8000字。

短文：有创造性的或包含新的见解，但不要求是完整的工作；对已有工作的较大改进和提高。字数不超过4000字。

技术报告：有重大或较大的实用、推广价值，技术上先进和内容完整的研究报告。字数不超过8000字。

简报：技术上先进，有一定的实用或推广价值的研究成果(包括阶段成果)的扼要报道。字数不超过3000字。

通讯：有关本刊已发表文章的问题讨论或简短表达的新思想和对学科发展的建议等。

综述：针对计算机科学技术的新兴或活跃领域，对国内外发展状况及趋势的全面系统的综合评述。

2.来稿一式三份，文章务必有二百字中文摘要和英文摘要(英文摘要放在中文摘要前面，中文题目、作者姓名及单位名称的后面。英文摘要包括题目，姓名，单位名称。英文摘要，不要另附，免得丢失)。

3.来稿务必做到清稿定稿，一经排版，不得再作任何文字上的修改。三份文稿均请用钢笔写在单面稿纸上，字迹清楚，稿面整洁。

4.文中的插图要精绘，图中文字书写清楚，插图在稿中所占的位置用方框标出，注明图号、图注。

5.文中的外文字母、符号必须分清大、小写，正、斜体；上、下角的字母、数码和符号，其位置高低应区别明显；容易混淆的外文字母，请在第一次出现时，用铅笔批清。文中需用黑体字之处，请在字下加波纹线。

6.参考文献只择最主要的列入，文献按文中出现先后次序编排，书写格式如下：

期刊：[编号]作者姓名，文章题目，刊物名称，卷数：期数(年份)，页数。

图书：[编号]作者姓名，书名，出版单位，地点，年份，页数。

四、反映重大成果的稿件希附有作者单位推荐信和审查意见。

五、刊登稿件按修改最后定稿时间排序。来稿一经发表，酌致稿酬，并赠送单行本30本。不拟刊登稿件，当妥为退还。

来稿请寄北京中国科学院软件研究所《软件学报》编辑部(邮政编码100080)(北京8718信箱)。