СТРУКТУРНЫЙ СИНТЕЗ ПРОГРАММ

Э.Х.Тыугу

Таллин, СССР

Описывается метод построения доказательства существования решения задачи и вывод из такого доказательства программы решения задачи, позволяющие практически синтезировать большие программы. Основой метода являются следующие две идеи уменьшения перебора при поиске доказательства:

для каждой задачи строится своя теория, в аксиомах которой свободные переменные заменены на константы, представляющие объекты данной задачи;

доказываются только структурные свойства вычислений, предполагая, что элементарные операторы выполняются правильно.

I. Введение

Существует по крайней мере четыре разных подхода к синтезу программ:

пошаговое усовершенствование программ;

синтез по примерам пар входных и выходных данных или по примерам вычислений;

синтез по описаниям данных;

синтез по доказательству теоремы о том, что искомое решение задачи существует.

Здесь рассматривается последний из перечисленных подходов, и описывается метод практического синтеза программ по описаниям задач, задаваемым на непроцедурном языке описаний - на проблемно-ориентированном языке (ПОЯ).

Мы рассмотрим синтез программ решения задач следующего вида: по заданному значению x , удовлетворяющему P(x), вычислить значение y , удовлетворяющее R(x,y). Здесь x и y переменные или конечные множества переменных - входные и выходные переменные программы. Q=(P(x),R(x,y)) выражает условия задачи, которые должны содержать достаточно информации

для синтеза нужной программы. Форма условий задачи зависит от техники синтеза. Также виды входов и выходов задачи могут сильно варьироваться. Например, мы можем захотеть построить программу для нахождения числа, составления таблицы, документа, чертежа, программы.

Каждой задаче (x,y,Q) соответствует теорема существования решения:

для каждого x , для которого P(x) , существует y такой, что R(x,y) . Истинность теоремы существования является необходимым, но не является достаточным условием для разрешимости задачи.

Путь от задачи к программе ее решения может быть описан следующей схемой:

задача → теорема → доказательство → программа.

Он содержит три основных этапа:

I. Составление формального описания задачи в виде теоремы;

2. Доказательство теоремы;

3. Извлечение программы из доказательства.

Если эти шаги выполняются автоматически, то человек может быть уверен в том, что он получит программу, которая точно соответствует описанию задачи. Но составление правильного описания задачи остается его заботой, и вместо отладки программ придется отлаживать описания задач. Это проще в том отношении, что нет необходимости учитывать многие детали вычислений, поскольку задача описывается содержательно.

Ситуацию в области синтеза программ путем доказательства теорем поясняет рис. I. Эта область частично принадлежит к программированию, а частично - к логике. Опыт программирования, накопленный за многие годы этой деятельности, сократил путь от задачи к программе (0). Продвигаясь от задачи к программе через теорему существования и ее доказательство, приходится проходить более длинный путь. Шаг (I) от задачи к теореме ведет из области программирования в область логики. Поиск доказательства теоремы (шаг 2) выполняется в рамках логики, иногда - далеко от программирования. Извлечение программы из доказательства (шаг 3) приносит нас опять в программирование. Этот путь можно сократить, разработав специальные

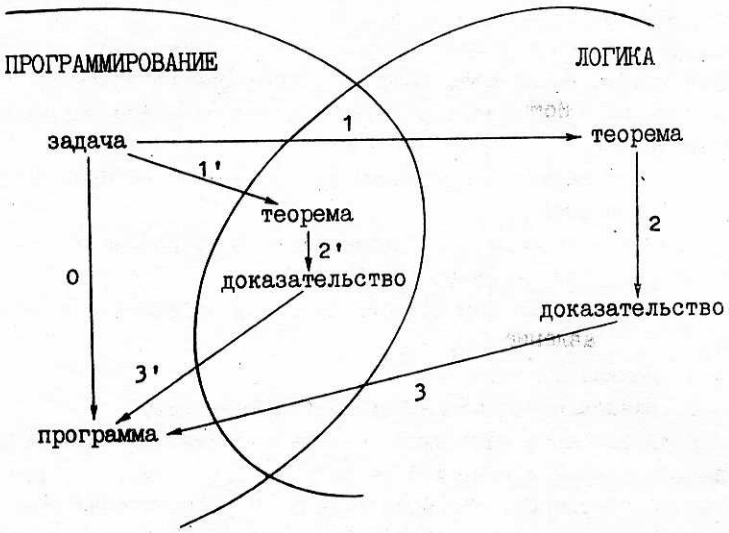

Рис. I.

Пути от задачи к программе.

теории для формулировки и доказательства теорем существования. Целью является:

перенести теоремы и доказательства из области чистой логики ближе к программированию, приблизив доказательства к программам;

упростить процесс поиска доказательства теорем путем учета особых свойств последних. Это позволит делать шаги 2' и 3' более короткими. Но различие между исходным описанием задачи и теоремой существования может не сокращаться, так как хорошо разработанные методы трансляции с формальных языков позволяют достаточно просто делать длинный шаг (I') от задачи к теореме.

При структурном синтезе программ для сокращения пути от задачи к программе использованы следующие две идеи:

для каждой задачи строится своя частная теория. Поскольку теперь эта теория требуется для доказательства только одной теоремы, то она может быть хорошо приспособлена для доказательства, соответствующего задаче. В частности, в аксиомах такой теории удается заменить большинство переменных на константы, соответствующие объектам данной задачи;

при доказательстве учитываются только структурные свойства вычислений. Правильность элементарных операций не доказывается. Предполагается, что если вычислимость некоторого объекта следует из аксиом, то будет вычислено именно правильное значение. В частности, отсюда следует, что при наличии различных путей вычисления, любой из них дает правильный результат.

Такое предположение может быть оправдано тем, что описание условий корректности элементарных шагов вычислений не проще (и не надежнее), чем описание самих операторов, выполняющих данные вычисления. На самом деле при таком подходе "элементарный шаг" вычислений может быть описан программой любой длины, как только правильность этой программы определена каким-либо способом заранее.

2. Вычислительная модель задачи

Рассмотрим теперь модель, содержащую все объекты, которые могут потребоваться при решении некоторой задачи, а также

функциональные отношения между этими объектами. Имея такую модель, мы сможем строить программу решения задачи, работающую только с объектами данной модели и содержащую только функции из нее.

Вычислительной моделью задачи назовем пару (X,F) . X - множество, содержащее объекты, вычисляемые хотя бы в одном процессе решения задачи, F - множество функциональных отношений, используемых для вычисления объектов из X . Элементы множества X соответствуют состояниям переменных строящейся программы. Например, если в программе содержится цикл, то X будет содержать последовательность объектов, возможно, бесконечную. Только если каждая переменная программы вычисляется не более одного раза, имеется взаимно-однозначное соответствие между переменными программы и элементами множества X . Если, кроме того, программа не содержит ветвлений и рекурсий, то вычислительная модель сама может рассматриваться как программа, вычисления в которой полностью управляются потоками информации.

Оказывается, что извлечь программу из вычислительной модели значительно проще, чем доказать теорему в какой-нибудь интуиционистской теории.

С другой стороны - вычислительную модель задачи удается очень часто непосредственно получить из описания задачи, заданного на ПОЯ. Семантика предложений на ПОЯ может быть представлена семантическими моделями, являющимися строительными блоками вычислительной модели. Этот подход был уже описан в [I] и является основой компилятора с языка УТОПИСТ [2]. Последний является расширяемым языком для построения ПОЯ.

Понятие вычислительной модели может быть определено другим способом как множество аксиом особого вида. Аксиомы содержат символы констант для объектов из X и функциональные символы для элементов F .

Аксиомы теории, описывающей класс задач, могут содержать переменные. Последние заменяются на константы при переводе конкретного описания задачи. Например, пусть для некоторого класса задач имеются аксиомы;

углы треугольника $(\alpha, \beta, \gamma) \Rightarrow (\forall\alpha\forall\beta\exists\gamma(\gamma = 180^{\circ} - \alpha - \beta))\&$

$$\&(\forall\beta\forall\gamma\ \exists\alpha(\alpha = 180^{\circ} - \beta - \gamma)\&$$
$$\&(\forall\gamma\forall\alpha\ \exists\beta(\beta = 180^{\circ} - \gamma - \alpha)$$

Если в условиях некоторой задачи определен конкретный треугольник T , то вместо приведенной аксиомы можно получить три новые, в которые вместо переменных α , β , γ входят константы α от T, β от T, γ от T:

$$\gamma \text{ от } T = 180^{\circ} - \alpha \text{ от } T - \beta \text{ от } T,$$
$$\alpha \text{ от } T = 180^{\circ} - \beta \text{ от } T - \gamma \text{ от } T,$$
$$\beta \text{ от } T = 180^{\circ} - \gamma \text{ от } T - \alpha \text{ от } T.$$

Стоит отметить, что из очень простого описания "T треугольник", данного в тексте задачи, можно извлечь много аксиом о вычислимости, подобных приведенным.

3. Предложения вычислимости

Определим теперь формулы для наших специальных теорий. Обозначим через $a \overset{f}{\longmapsto} b$ высказывание, что из a вычислимо b применением функции f , где a,b - объекты или конечные множества объектов. Результат вычислений определен однозначно функцией $f=\langle f_1,\ldots,f_n\rangle$ так, что $b_i=f_i(a)$ для любого b_i из b . В частности, если x,y - переменные,то $x \overset{f}{\longmapsto} y \rightleftharpoons (\forall x \exists y(y=f(x))$, и f - реализуемая программой функция.Назовем формулу $a \overset{f}{\longmapsto} b$ отношением вычислимости.

Вычислимость объектов может зависеть от условий, выраженных некоторой формулой Г . Поэтому допустим формулы со следующей схемой $\Gamma \Longrightarrow a \overset{f}{\longmapsto} b$ и будем называть их предложениями вычислимости. Г выражает условия вычислимости.

Предложения вычислимости полезны нам следующим образом:

вычислительная модель может быть представлена как множество отношений вычислимости, или предложений вычислимости с тождественно истинными посылками;

теорема существования решения выражается через отношение вычислимости следующим образом:

$$\exists f(x \overset{f}{\longmapsto} y)\ ;$$

любую совокупность предложений вычислимости можно рассматривать как систему аксиом некоторой теории, в которой следует искать доказательство теоремы существования. В зависимости от вида условий вычислимости в аксиомах применимы

разные стратегии поиска, которые будут рассмотрены в следующем параграфе.

Теорема существования не является формулой I-го порядка. Но применив правило вывода

$$0^\circ \quad \frac{A(f_o)}{\exists f\, A(f)},$$

можно эту формулу доказать, как только доказана формула $x \overset{f_o}{\vdash\!\!\rightarrow} y$ для некоторого f_o. Мы будем искать вывод $x \overset{f_o}{\vdash\!\!\rightarrow} y$ из множества аксиом, заданных в виде предложений вычислимости.

4. Метод структурного синтеза

4.I. Синтез программ по информационным связям

Отношения вычислимости позволяют представить любые информационные связи между конечным множеством объектов. Если для решения задачи ни один объект не надо вычислять более одного раза, можно на вычислительной модели задачи определить порядок, который гарантирует возможность вычислений, если функции применять в заданном порядке. Идея заключается в том, что если известно, что $a \overset{f_1}{\vdash\!\!\rightarrow} b$ и $b \overset{f_2}{\vdash\!\!\rightarrow} c$, то последовательное применение f_1 и f_2 дает c из a , то есть можно вывести новое отношение вычислимости:

$$a \overset{(f_1;f_2)}{\vdash\!\!\rightarrow} c.$$

Пусть x,y,... - конечные наборы объектов (констант или переменных). Тогда вычислимость выражается следующими правилами вывода.

$$1^\circ \quad \frac{y \subseteq x}{x \overset{S_{x,y}}{\vdash\!\!\rightarrow} x'},$$

где $S_{x,y}$ - селектор-функция, выбирающая значения переменных y из заданных значений переменных x .

$$2^\circ \quad \frac{x \overset{f_1}{\vdash\!\!\rightarrow} y,\ x \overset{f_2}{\vdash\!\!\rightarrow} z,\ w=y \cup z}{x \overset{(f_1,f_2)}{\vdash\!\!\rightarrow} z},$$

где (f_1,f_2) обозначает совместное выполнение функций f_1 и f_2.

$$3^\circ \quad \frac{x \xrightarrow{f_1} y,\ y \xrightarrow{f_2} z}{x \xrightarrow{(f_1;f_2)} z},$$

где $(f_1;f_2)$ обозначает последовательное выполнение функций f_1 и f_2.

Если для заданной задачи (x,y,Q), где Q - вычислительная модель с тождественно-истинными условиями вычислимости, формула $x \xrightarrow{f} y$ выводима в теории с системой аксиом Q и правилами вывода 1°, 2°, 3°, тогда задача (x,y,Q) разрешима и программа ее решения строится из функций, входящих в аксиомы Q.

Правило вывода 2° можно заменить на правило $2'$ в предположении, что вычислимые значения объектов не зависят от порядка применения функции. (Но последнее предположение следует из предположения, сделанного нами ранее, что по аксиомам допустимы только правильные вычисления.).

$$2' \quad \frac{x \xrightarrow{f_1} y,\ x \xrightarrow{f_2} z,\ w=yUz}{x \xrightarrow{(f_1;f_2)} w}.$$

Поскольку мы далее ограничимся рассмотрением синтеза последовательных программ, то будем пользоваться как раз правилом $2'$. Подробное описание синтеза асинхронных программ на вычислительных моделях описано в [3].

4.2. Обработка последовательностей

Пусть $x_1, x_2, \ldots$ - последовательность объектов, на которой заданы отношения вычислимости $u \xrightarrow{f_0} x_1$, $x_i \xrightarrow{f} x_{i+1}$ для $i = 1,2,\ldots$ n-кратное применение правила 3° дает программу вычисления n-го элемента последовательности:

$$f_0; \underbrace{f; \ldots ; f}_{n \text{ раз}}$$

Эту же программу можно свернуть в цикл Φ_0; for i to n do Φ od, где Φ_0; Φ - операторы, выполняющие вычисления по функциям f_0 и f.

Задача становится более сложной, если требуется вычислить x_i, для которого $\neg P(x_i)$, и $P(x_j)$, если $j < i$. Соответствующая программа может быть Φ_0; while $P(x)$ do Φ od, где x - переменная, которой на каждом шаге присваивается вычисленное

значение x_i. О полученной программе, вообще говоря, не известно, завершит ли она работу. Доказательство завершения следует проводить отдельно.

Синтез циклов для обработки последовательностей был описан в 1958 г. [4]. Более общие результаты приведены в [5]. Там рассматривается n последовательностей

$$x_{s,1}, x_{s,2}, \ldots \qquad s=1,2,\ldots,n.$$

Предполагается, что для некоторых фиксированных $m_1^\circ, \ldots, m_n^\circ$ значения объектов $x_{s,i}$, $i \le m_s^\circ$ заданы. Ищутся значения объектов $x_{1,m_1}, x_{2,m_2}, \ldots, x_{n,m_n}$ для фиксированных $m_1, m_2, \ldots, m_n$. Отношения вычислимости имеют вид

$$\{x_{1,i-\Delta(s,1)}, \ldots, x_{n,i-\Delta(s,n)}\} \xmapsto{f_s} x_{s,i},$$

где $0 \le \Delta(s,j) \le m_j^\circ$. В [5] доказано, что для разрешимой проблемы существует конечная последовательность функций f_s, которая вычисляет по одному новому объекту $x_{s,i}$ в каждой последовательности, используя только объекты $x_{s,i}$, $j \le i-\Delta(s,j)$. Программа, вычисляющая по этой последовательности функций, может быть принята за тело цикла, решающего данную задачу.

4.3. Синтез ветвящихся программ

Пусть условия вычислимости Г в аксиомах являются вычислимыми предикатами $P_1, P_2, \ldots$, которые вычисляются программами $p_1, p_2, \ldots$

Правило вывода

$$4^\circ \qquad \frac{P_1(w) \vee \ldots \vee P_k(w),\ P_1(w) \Rightarrow x \xmapsto{f_1} y, \ldots x \xmapsto{f_k} y}{x\ U w \xmapsto{f} y}$$

позволяет получить ветвящуюся программу

$$f = \underline{\text{if}}\ p_1(w)\ \underline{\text{then}}\ f_1\ \underline{\text{elif}}\ p_2(w)\ \underline{\text{then}}\ \ldots\ \underline{\text{else}}\ f_k\ \underline{\text{fi}}$$

Применяя правило 4° вместе с правилами 1°, 2' и 3°, получим программы, составленные из линейных участков и условных предложений. Если нет необходимости доказывать $P_1(w) \vee \ldots \vee P_k(w)$ (например, если истинность этой формулы следует из некоторых общих соображений о задачах), то может применяться следующая очень простая стратегия поиска доказательства. Сперва проверяются все аксиомы с тождественно истинными условиями вычислимости и используются все применимые из них.

Затем учитываются аксиомы $P_i(w) \Rightarrow x \vdash\!\!\xrightarrow{f_i} y$, для которых вычислены $w_i x$ и генерируется условное предложение if ... fi. Каждая ветвь условного предложения синтезируется рекурсивным применением этой же самой стратегии.

В более общем случае условия вычислимости в аксиомах могут быть нормальными формулами, определенными в [6]. В этом случае форма синтезируемой программы остается такой же, как в рассмотренном случае, но поиск доказательства усложняется, поскольку надо доказывать невычислимые части нормальных формул. Доказательство истинности $P_1(w) \vee \ldots \vee P_k(w)$ может возлагаться на программиста. В некоторых специальных случаях, например, при переводе таблиц решений, доказательство этой формулы получается автоматически тоже достаточно просто.

Если в качестве результата синтеза допустить частичные программы, то нет необходимости доказывать формулу $P_1(w) \vee \ldots \vee P_k(w)$. Обозначим $a \vdash\!\!\xrightarrow{f} b$, что существует частичная функция f для вычисления b из a. Тогда правило 4° может быть заменено на

$$4' \qquad \frac{P_1(w) \Rightarrow x \vdash\!\!\xrightarrow{f_1} y, \ \ldots \ P_k(w) \Rightarrow x \vdash\!\!\xrightarrow{f_k} y}{x \ w \vdash\!\!\xrightarrow{f} y},$$

где f = if $P_1(w)$ then f_1 elif ... ... elif $P_k(w)$ then f_k else failure fi.

failure является процедурой, сообщающей о неразрешимости задачи и завершающей выполнение программы.

4.4. Синтез процедур

Практически весьма полезно выделить при решении задачи как можно больше подзадач. Таким способом задача делится на части, которые с большой вероятностью будут проще самой задачи, и, следовательно, синтез программ их решения будет тоже проще. Выделение подзадач может произойти уже при задании аксиом. В частности, можно поставить применимость некоторых аксиом в зависимость от разрешимости подзадач. Например, чтобы вычислить значение определенного интеграла $z = \int_0^x y \, du$, необходимо решить подзадачу вычисления y по u. Это может быть выражено следующим предложением вычислимости

$$\exists f (u \vdash\!\!\xrightarrow{f} y) \Rightarrow (x \vdash\!\!\xrightarrow{h} z),$$

где $h=H(f_o)$ реализована некоторой подпрограммой численного интегрирования, имеющей в качестве одного параметра программу вычисления f . Доказательство того, что подзадача разрешима, дает программу вычисления функции f_o.

Здесь условие вычислимости содержит связанную функциональную переменную f , следовательно, такие аксиомы уже не выразимы в теориях первого порядка. Все же существует эффективный метод поиска доказательства разрешимости, если заранее известно, что ни одно предложение вычислимости не надо применять в пределах доказательства разрешимости подзадачи более одного раза. Для каждой подзадачи доказательство строится уже рассмотренными методами, если предположить, что условия вычислимости вида $\exists f(u \xrightarrow{f} y)$ истинны. Но для окончательного доказательства необходимо провести проверку разрешимости подзадач на u - или дереве подзадач.

Следует отметить, что тело цикла можно синтезировать как программу решения подзадачи. Управляющие структуры циклов, соответствующих разным схемам индукции, могут быть заранее запрограммированы и представлены в виде аксиом. Именно так происходит синтез циклов (и рекурсивных программ) в системе программирования ПРИЗ [7].

5. Примеры применений

5.I. Синтез программ обработки данных

Обработка деревьев. Объекты, связанные структурными отношениями вида "x состоит из компонент $x_1,\ldots,x_k$ ", где любая компонента x_i может опять быть составной, образуют дерево. Каждому структурному отношению соответствуют отношения вычислимости: конструктор $\{x_1,\ldots,x_k\} \xrightarrow{constr} x$ и селекторы $x \xrightarrow{select_i} x_i$, $i = 1,2,\ldots,k$. Эти отношения вычислимости полностью описывают возможность вычислений, определенную структурными свойствами данных.

Пользуясь только правилами вывода I°, 2° и 3° и самым простым методом синтеза из п. 4, возможно получать все те программы вычисления одних объектов по другим, которые следуют из определений данных.

Обработка семантических сетей. К структурным отношениям могут добавляться и другие отношения вычислимости между объ-

ектами, задаваемые, например, уравнениями. Синтез программ на сетях из объектов и отношений вычислимости был описан в [7] и для безусловных предложений вычислимости опять выполним применением только правил I°, 2°, 3°.

Вот примеры описаний более сложных структур данных на языке УТОПИСТ [2], в трансляторе которого используется синтез программ:

```
LET' TRIANGLE: (A,B,C,ALPHA,BETA,GAMMA: REAL';
        EQN' ALPHA + BETA + GAMMA = 180;
        A/SIN' ALPHA = B/SIN' BETA;
                    ⋮ );
T1: TRIANGLE A=1.5, ALPHA=60, BETA=45;
T2: TRIANGLE A=T1.B, ALPHA=T1.BETA; BETA=60;
```

Пример разрешимой задачи с условиями в виде приведенных выше описаний следующий: COMPUTE' T2.B.

Сказанное относительно синтеза программ на семантических сетях справедливо также для систем управления базами данных, в которых синтез программ обработки запросов производится по схеме базы данных. В системе управления базами данных DABU метод структурного синтеза прямо применен как для обработки запросов, так и для манипулирования данными [8].

Последовательная обработка данных. Последовательный файл можно рассматривать либо как последовательность записей u_1, u_2,... , либо как последовательность состояний файла x_0, x_1, x_2,..., связанных между собой отношениями вычислимости:

$$\left.\begin{array}{l} x_i \xrightarrow[get]{} u_i \\ x_{i-1} \xrightarrow[next]{} x_i \\ \{x_{i-1}, u_i\} \xrightarrow[put]{} x_i \end{array}\right\} \quad i=1,2,\ldots$$

(Макрокоманда GET , применяемая в программировании, чаще всего является последовательностью функций get и next).

Весьма просто представить задачу обработки последовательных файлов как задачу обработки последовательностей, рассмотренную в п. 4, и решать ее соответственным образом.

Например, пусть имеются входной файл $u_{11}, u_{12},\ldots,$ и выходной файл $u_{21}, u_{22},\ldots$. Очевидно, должны быть заданы и

некоторые отношения вычислимости для получения записей выходного файла по записям входного, например,

$$u_{1i} \overset{f}{\longmapsto} u_{2i}, \quad i=1,2,\ldots$$

Эти отношения вместе с универсальными отношениями последовательных файлов:

$$\left.\begin{array}{l} x_{1,i-1} \overset{next}{\longmapsto} x_{1,i} \\ x_{1,i} \overset{get}{\longmapsto} u_i \\ \{x_{2,i-1}, u_{2i}\} \overset{put}{\longmapsto} x_{2,i} \end{array}\right\} \quad i=1,2,\ldots$$

дают вычислительную модель, на которой разрешима задача получения состояния выходного файла для любого заданного i.

Функция f, как правило, сама получается путем синтеза из отношений, связывающих компоненты записей входного и выходного файла. Для этого применимо все, сказанное в начале данного параграфа.

Конструирование программ обработки последовательных файлов (не автоматически) рассмотрено подробно в книге [9]. Этот подход автоматизации программирования практически применяется в генераторах программ отчетов, нашедших на практике широкое применение.

5.2. Реализация атрибутной техники

Интересным примером практического применения синтеза программ является реализация семантики описательных языков. Этот подход может быть обобщен так, что семантическую часть компилятора можно строить по формальному описанию грамматики языка.

Пусть G - атрибутная грамматика [10] с набором P порождающих правил, и набором X атрибутов. Каждому правилу $p \in P$ сопоставлено множество отношений вычислимости над вхождениями атрибутов в правило p. Эти отношения выражают вычислимость атрибутов и вместе с программами функций $f,\ldots,g$, которые входят в отношения вычислимости, определяют семантику языка L, описанного грамматикой G.

Допустим, что G корректная атрибутная грамматика языка L

и prog является атрибутом, значением которого является машинный код исходной программы. В [II] показан способ построения вычислительной модели по заданному дереву разбора текста программы так, что задача $\exists f(\emptyset \vdash_f prog)$ будет разрешимой на вычислительной модели. Такая вычислительная модель будет содержать те отношения вычислимости, которые соответствуют порождающим правилам, входящим в дерево разбора. В [II] описаны эксперименты синтеза программ по атрибутной грамматике и дереву разбора, представленному на языке УТОПИСТ. Транслятор с УТОПИСТа, который и выполняет синтез программ, был использован в данном случае как динамический вычислитель атрибутов. Дальнейшее развитие данного метода возможно применением техники визитов для вычисления атрибутов. Описывая визиты как подзадачи, можно надеяться реализовать статическую планировку вычисления атрибутов. Таким образом можно попытаться при помощи транслятора с УТОПИСТа полностью автоматизировать построение семантической части компилятора по атрибутной грамматике.

литература

I. Tyugu E.H. Data base and problem solver for computer aided design. Information Processing 71, North-Holland Publ. Co., Amsterdam, 1972.

2. Мяннисалу М.А. и др. Язык УТОПИСТ, "Алгоритмы и организация обработки экономической информации". - М.: Статистика, 1977, с. 80-118.

3. Плакс Т.П. Синтез параллельных программ на вычислительных моделях. - Программирование, 1977, № 4, с. 55-63.

4. Любимский Э.З. Автоматизация программирования и метод программирующих программ: Автореф. дис. на соиск. учен. степени канд. физ.-мат. наук. - ИПМ АН СССР, М., 1968.

5. Задыхайло И.Б. Составление циклов по параметрическим записям специального вида, Журн. вычисл. математики и мат. физики. 1963, т. 3, № 2. с. 337-357.

6. Непейвода Н.Н. О построении правильных программ. - Вопр. кибернетики, 1978, т. 46, с. 88-122.

7. Tyugu E.H. A programming system with automatic program synthesis. Lecture Notes in Computer Science, v.47, Methods of Algor. Language Implementation, Springer-Verlag, Berlin, 1977, p. 251-267.
8. Калья А.П., Мацкин М.Б. Интеллигентный диалог с банками данных. - Тр. сов.-фин. симпоз. по интерактивным системам, Тбилиси, 1979, ч. I, с. 124-136.
9. Jackson M.A. Principles of program design. Academic Press, London, N.Y., 1975.
10. Knuth D. Semantics of context-free languages. Mathem. System. Theory, v.2, no.2, 1968, p. 127-144.
11. Пеньям Я. Метод автоматической реализации семантики в компиляторах. - Кибернетика, 1980, № 2, с. 36-41.