4. Palaniappan S. Ling C. Clinical Decision Support Using OLAP With Data Mining // Intern. J. of Computer Science and Network Security. 2008. Vol. 8. № 9. P. 290–296.

5. Коробко А. В., Пенькова Т. Г. Метод концептуального OLAP-моделирования на основе формального концептуального анализа // Вестник СибГАУ. 2010. № 4 (30). С. 74–79.

6. Krajca P., Outrata J., Vychodil V. Parallel Recursive Algorithm for FCA // Proc. of the Sixth Intern. Conf. on Concept Lattice and their Applicatons. Olomouc, 2008. P. 71–82.

7. Lindig C. Fast concept analysis // Proc. of the Intern. Conf. on Conceptual Structures (ICCS). Aachen : Shaker Verlag, 2000.

8. Concept Lattices : Second Intern. Conf. on Formal Concept Analysis / P. Eklund. Sydney, 2004. P. 23–26.

9. Kuznetsov S. O., Obiedkov S. A. Comparing Performance of Algorithms for Generating Concept Lattices // J. of Experimental and Theoretical Intelligence. 2002. Vol. 14. P. 189–216.

10. Korobko A., Penkova T. On-line analytical processing based on Formal concept analysis // Procedia Computer Science. 2010. Vol. 1. № 1. P. 2305–2311.

11. Penkova T.G., Korobko A.V. Constructing the integral OLAP-model based on Formal Concept Analysis // Proc. of the 34th Intern. Convention. Opatija, 2011. P. 225–229.

12. Wille R. Restructuring Lattice Theory: an approach based on hierarchies of concept. Dordrecht ; Boston : Reidel, 1982. P. 445–70.

A. V. Korobko, T. G. Penkova

ALGORITHMS OF COMPOSITION OF THE INTEGRAL OLAP-MODEL IN OBJECT DOMAIN

The algorithms of composition of an integral OLAP-model based on the search of cube-concepts and composition of concept lattice of OLAP- cubes are described. Suggested algorithms are supplemented for integral OLAP- model of scientific activity of an organization.

Keywords: integral OLAP-model, on-line analytical data processing, formal conceptual analysis.

УДК 681.142.2

В. А. Кравченко, П. Б. Могнонов, Д. Н. Чимитов

ПРЕДСТАВЛЕНИЕ ЗНАНИЙ В ФУНКЦИОНАЛЬНЫХ ГРАММАТИКАХ

Рассмотрен один из способов представления знаний (теорий) с помощью функциональных грамматик. Сделана попытка дать единую схему решения задачи путем ее погружения в функциональную грамматику теории. Показано, что вычисления на уровне типов с помощью функциональных грамматик имеют существенное значение для синтеза программ.

Ключевые слова: контекстно-ориентированное программирование, объектно-ориентированное программирование, структура данных.

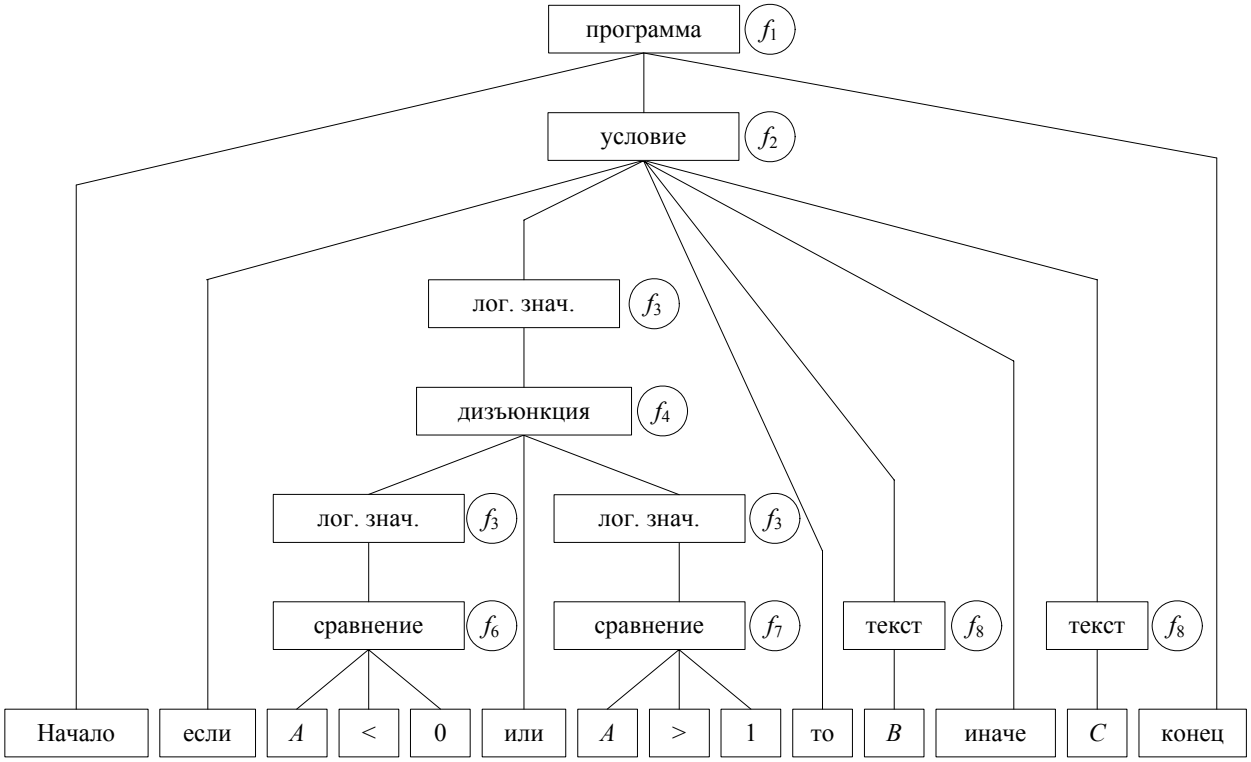
Рассмотрим вопрос о формализации языка с точки зрения формирования понятий на основе анализа отношений между различными объектами.

В программировании существенную роль играет понятие полиморфизма. В данной статье нам хотелось бы выдвинуть следующий тезис: любая предметно-ориентированная теория (например, электротехника, механика и т. п.) является по существу полиморфной программой, которая написана на специализированном естественном языке, и если использовать функциональные грамматики [1; 2], то можно описать практически любую теорию (группу теорий) в программной оболочке, способной генерировать решение любой задачи в рамках данной теории (групп теорий).

Согласно данному подходу, любое предложение естественного языка, а также любая программа на языке программирования может быть представлена либо математически в виде суперпозиции функций (рис. 1), либо графически в виде дерева синтаксического разбора с функциями в узлах (рис. 2).

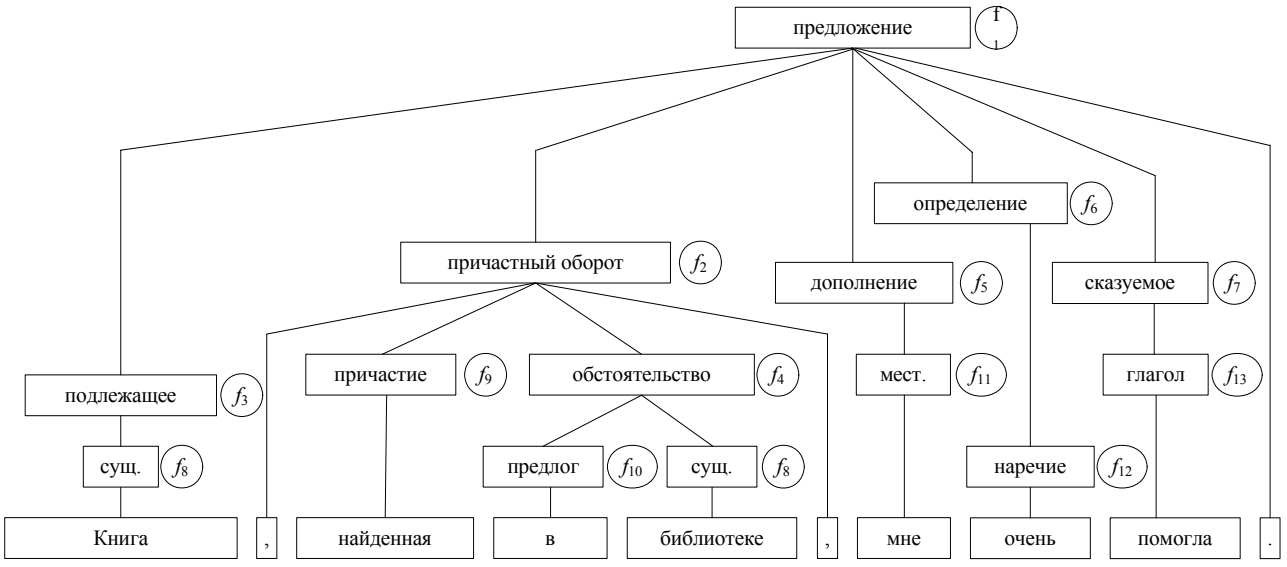
При этом оба представления являются эквивалентными и выражают одни и те же зависимости элементов текста.

Однако суперпозиции функций, представленные на рис. 1 и 2, являются неполными, так как они выражают лишь разбор на уровне синтаксиса. Ниже мы будем рассматривать вычисления, основанные на функциональных грамматиках, и процессы функционально-логических вычислений на иерархическом уровне типов, т. е. вычисления на уровне предметной и общей логики типов для построения суперпозиций функций, или генерации процедур.



$$? = f_1(f_2(f_3(f_4(f_3(f_6(A,0)),f_3(f_7(A,1))),f_8(B),f_8(C)))$$

Рис. 1. Дерево синтаксического анализа и суперпозиция функций для программы



$$? = f_1(f_3(f_8(\text{книга})),f_2(f_9(\text{найденная}),f_4(f_{10}(\text{в}),f_8(\text{библиотеке}))),f_5(f_{11}(\text{мне})),f_6(f_{12}(\text{очень})),f_7(f_{13}(\text{помогла})))$$

Рис. 2. Дерево синтаксического анализа и суперпозиция функций для предложения русского языка

Однако для выполнения вычислений на уровне логики типов необходимо развить саму общую теорию логики типов, или общую теорию понятий, которая существенно отличается от математической теории: общая теория логики типов (понятий) предполагает разработку специального языка (или семейств языков) для представлений знаний об окружающем нас мире, в то время как математическая логика опирается на группу своих языков логики, например на язык исчисления предикатов и теорию рекурсивных функций,

которые до настоящего времени были ориентированы на решение только математических задач. Однако в связи с развитием информатики все более актуальной становится относительно недавно возникшая инженерия знаний.

Так, дерево на рис. 1 можно углубить, расписав в качестве суперпозиций элементы *A*, *B* и *C*. Во втором же случае (см. рис. 2) представлению в виде суперпозиций можно подвергнуть каждое слово, выразив его значение через другие слова. Но в каждом языке в результате самого глубокого разложения можно дойти до тех функций, которые не подлежат дальнейшему функциональному представлению и в этом смысле являются элементарными. Такие функции будем называть *базисными*.

Пример 1. Приведем простейшую функциональную грамматику, описывающую основные законы кинематики. Напомним, что главными объектами (типами) в этом разделе физики являются расстояние, время, скорость и ускорение. Между ними существуют следующие известные соотношения (предикаты):

$$\left\{ V = V_0 + a \cdot t; \quad S = V_0 \cdot t + \frac{a \cdot t^2}{2}; \quad S = \frac{V^2 - V_0^2}{2 \cdot a} \right\}. \quad (1)$$

Из предиката (1) порождаются 12 функциональных зависимостей:

$$V = V_0 + a \cdot t,$$
$$V = \sqrt{2 \cdot a \cdot S + V_0^2},$$
$$V_0 = V - a \cdot t,$$
$$V_0 = \frac{S}{t} - \frac{a \cdot t}{2},$$
$$V_0 = \sqrt{V^2 - 2 \cdot a \cdot S},$$
$$S = V_0 \cdot t + \frac{a \cdot t^2}{2} \text{ и т. д.}$$

Дадим обычно принятые в функциональных грамматиках определения объектов (типов, понятий):

– <Speed> → <SpeedInKM/H> | <SpeedInM/S> | <SpeedInKM/S>;

– <Start> → <StartInKM/H> | <StartInM/S> | <StartInKM/S>;

– <Dist> → <DistInKM> | <DistInM> | <DistInSM>;

– <Time> → <TimeInH> | <TimeInMIN> | <TimeInS>;

– <Acc> → <AccInKM/HH> | <AcctInM/SS>;

– <SpeedInKM/H> → (R: real; км/ч);

– <SpeedInM/S> → (R: real; м/с);

– <SpeedInKM/S> → (R: real; км/с);

– <StartInKM/H> → (R: real; км/ч);

– <StartInM/S> → (R: real; м/с);

– <StartInKM/S> → (R: real; км/с);

– <DistInKM> → (R: real; км);

– <DistInM> → (R: real; м);

– <DistInSM> → (R: real; см);

– <TimeInH> → (R: real; ч);

– <TimeInMIN> → (R: real; мин);

– <TimeInS> → (R: real; с);

– <AccInKM/HH> → (R: real; км/ч^2);

– <AccInM/SS> → (R: real; м/с^2).

Условно обозначим приведенные выше продукции через $G_T^{(0)}$.

Для заданной грамматики между нетерминальными понятиями имеются следующие соотношения (<Предикаты>):

– <Speed> = <Start> + <Acc>*<Time>;

– <Dist> = <Start>*<Time> + <Acc>*<Time>*<Time>/2;

– <Dist> = [<Speed>*<Speed> – <Start>*<Start>] / (2*<Acc>).

Программная оболочка по известным алгоритмам должна автоматически порождать из указанного предиката следующие функции (<Автоматически порожденные функции>):

– <Speed> → <Start> + <Acc>*<Time> {f_1};

– <Speed> → КОРЕНЬ [2*<Acc>*<Dict> + <Start>*<Start>] {f_2};

– <Start> → <Speed> – <Acc>*<Time> {f_3};

– <Start> → <Dist>/<Time> – <Acc>*<Time>/2 {f_4};

– <Start> → КОРЕНЬ [<Speed>*<Speed> – 2*<Acc>*<Dict>] {f_5};

– <Dist> → <Start>*<Time> + <Acc>*<Time>*<Time>/2 {f_6};

– <Dist> → [<Speed>*<Speed> – <Start>*<Start>] / [2*<Acc>] {f_7};

– <Time> → [<Speed> – <Start>] / <Acc> {f_8};

– <Time> → [КОРЕНЬ [<Start>*<Start> + 2*<Acc>*<Dist>] – <Start>] / <Acc> {f_9};

– <Acc> → [<Speed> – <Start>] / <Time> {f_{10}};

– <Acc> → 2* [<Dist> – <Start>*<Time>] / [<Time>*<Time>] {f_{11}};

– <Acc> → [<Speed>*<Speed> – <Start>*<Start>] / [2*<Dict>] {f_{12}}.

Также имеются следующие функции преобразования единиц измерения объектов:

– g_1 = (<SpeedInKM/H>) <SpeedInM/S>: (0.278*R.<SpeedInKM/H>; м/с);

– g_2 = (<TimeInH>) <TimeInS>: (3600*R.<TimeInH>; с) и т.д.

Рассмотренный выше процесс образования функциональной грамматики теории G_T можно представить в виде следующей обобщенной схемы:

$$G_T^{(0)} + \text{<Предикаты>} = = > G_T^{(1)} + \text{<Автоматически порожденные функции>} = = > G_T.$$

В заданной функциональной грамматике рассмотрим интерпретацию конкретной задачи. Дано: *A*: <Acc> = (5.0; м/с^2); *S*: <Dist> = (6; км); *V*0: <Start> = (0.0; м/с). Необходимо найти: *T*: <Time> = (?.?; с).

Сформулируем процесс решения задачи программной оболочкой, которая загружена функциональной грамматикой G_T.

Шаг 1. Примем за аксиому величину, которую нужно найти, а за терминальные символы – заданные величины (в смысле грамматики).

Шаг 2. Выберем из грамматики G_T продукции, содержащие аксиому в левой части.

Шаг 3. Может существовать три ситуации: первая – в правых частях полученных продукций отсутствуют нетерминалы и тогда решением задачи будет путь в дереве от аксиомы к заданному данными продукциями узлу; вторая – в правых частях есть нетерминалы, тогда выбираем продукции для самого левого нетерминала; третья – в правых частях присутствует аксиома и такие ветви будем отбрасывать.

Шаг 4. Процесс повторяем до тех пор, пока не будет найден узел, содержащий только терминалы, т. е. решение задачи.

Таким образом, решение рассматриваемой нами задачи имеет вид суперпозиции функций, которая получается при чтении дерева разбора по пути от аксиомы до найденного узла, при этом функции, входящие в суперпозицию, находятся на ветвях дерева разбора, т. е. решение может быть представлено в виде дерева и соответствующей суперпозиции функций (рис. 3).

В примере 1 мы показали, что, используя функциональные грамматики, можно описать практически любую теорию (группу теорий) в программной оболочке, способной генерировать решение любой задачи в рамках данной теории (групп теорий). Процесс решения задачи по своей сути является построением подграмматики функциональной грамматики теории.

Процесс образования функциональной грамматики теории G_T можно представить в виде следующей обобщенной схемы:

$$G_T^{(0)} + \text{<Предикаты>} = = > G_T^{(1)} + \text{<Автоматически порожденные функции>} = = > G_T,$$

где $G_T^{(0)}$ – начальная грамматика, в которой приводятся определения понятий в термах: <Предикаты> описывают соотношения и связи между понятиями теории.

Объединение $G_T^{(0)}$ и <Предикаты> образует грамматику $G_T^{(1)}$, которая является входным интерфейсом пользователя, настраивающего систему на определенную теорию (группу теорий). Сама же система автоматически строит функциональную грамматику теории (группы теорий) внутри себя и готова к автоматическому решению задач.

Пример 2. Покажем, как, используя функциональную грамматику, можно создать модель состояние памяти для следующего программного фрагмента:

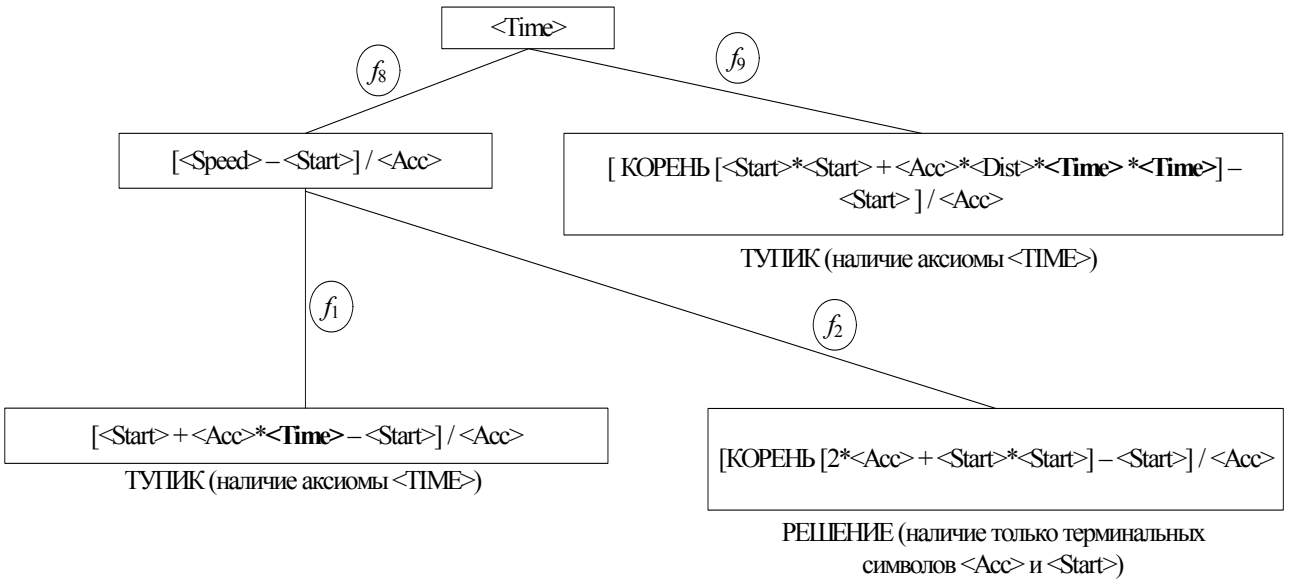
начало цел *A*, *B*, *C*, вещ *D*, *E*, … конец.

Цель этого примера состоит в том, чтобы продемонстрировать, как описания переменных *X*, *Y*, *Z*, *T*, *L* могут быть формально представлены в виде процесса управления памятью. Для этого введем формальные определения модели памяти, модели типов переменных и синтаксис описания переменных в языке.

Грамматика G_{M-1} – это грамматика описания общей памяти:

$$E \to M\ \omega\ F;\ M \to \text{вид}\ X\#,\ M \to \varepsilon;$$
$$F \to B\ \#\ B;\ B \to \varepsilon.$$

Здесь *E* – память; *M* – вспомогательная модель памяти для суперпозиции функций; *F* – внешняя память; *B* – файл ввода; ε – пустое множество.



$$\text{<Time>} = f_8(f_2(\text{<Acc>}, \text{<Start>}), \text{<Start>}, \text{<Acc>})$$

Рис. 3. Решение задачи в виде дерева разбора и суперпозиции функций

Начальное состояние памяти:

$$E \equiv \omega\ B\ \#.$$

Грамматика G_{M1} – это грамматика описания видов:

ВИД → цел, ВИД → вещ, ВИД → имя ВИД.

Упорядоченность

$$g_1 = (\text{имя ВИД})\ \text{ВИД}.$$

Грамматика G_0 – это грамматика синтаксиса:

<программа> → начало <описания>, <операторы> конец $\{f_1\}$;

<описания> → <описание> $\{f_2\}$; <описания> → <описания>, <описание> $\{f_3\}$;

<описание> → <описание>, <идент> $\{f_5\}$; <описание> → ВИД, <идент> $\{f_6\}$;

<идент> → X $\{f_{18}\}$; X → <буква>; X → X <буква>.

Функции f_i:

– f_2 = (конт t, знач x) конт: $(E, M)\ f_3(E, \varepsilon, M)$;

– f_3 = (конт t, знач x, y) конт: $((M\ \text{ВИД}\ X\ \#\ E, \varepsilon, \text{ВИД}^{(1)}\ X\ \#\ M) \perp\ |\ (E, \varepsilon, \text{ВИД}\ X\ \#\ M)\ f_3(\text{имя ВИД}\ X\ \#\ E, \varepsilon, M)\ |\ (E, \varepsilon, \varepsilon)\ \text{знач}\ (\varepsilon))$;

– f_5 = (знач x, текст y) знач: $(\text{ВИД}\ X\ \#\ M, X^{(1)})\ \text{ВИД}\ X\ \#\ M\ \text{ВИД}\ X^{(1)}\ \#$;

– f_6 = (текст x, y) знач: (ВИД, X) имя ВИД X #;

– f_{18} = (конт t, текст x) знач: $(M\ \text{ВИД}\ X\ \#\ E, X)$ ВИД.

Согласно введенной для заданного фрагмента программы грамматике можно построить дерево синтаксического разбора (рис. 4).

Результат представленного выше фрагмента программы будет определяться работой функций (или нетерминалов), находящихся в узлах дерева. Рассмотрим принцип работы функций на примере функции

$$f_6 = (\text{текст } x, y)\ \text{знач: (ВИД, } X)\ \text{имя ВИД } X\ \#.$$

Эта функция состоит из двух частей. Часть функции до двоеточия называется *заголовком функции*, после двоеточия – *телом функции*. Заголовок функции f_6 сообщает о том, что она имеет два аргумента: x и y, причем тип этих аргументов – текстовый. Результатом функции является тип знач.

Следует отметить, что любой аргумент, равно как и результат функции, может иметь один из четырех типов: текст, функ, знач, конт. Тип текст говорит о том, что аргумент необходимо рассматривать как последовательность символов (как терминальных, так и нетерминальных), тип функ – как невыполненную суперпозицию функций, тип знач – как значение выполненной суперпозиции, которая, как правило, передается по соответствующей исходящей ветви дерева разбора в качестве аргумента к вышестоящему нетерминалу. Тип конт применяется, когда параметр изменяет значение общей памяти.

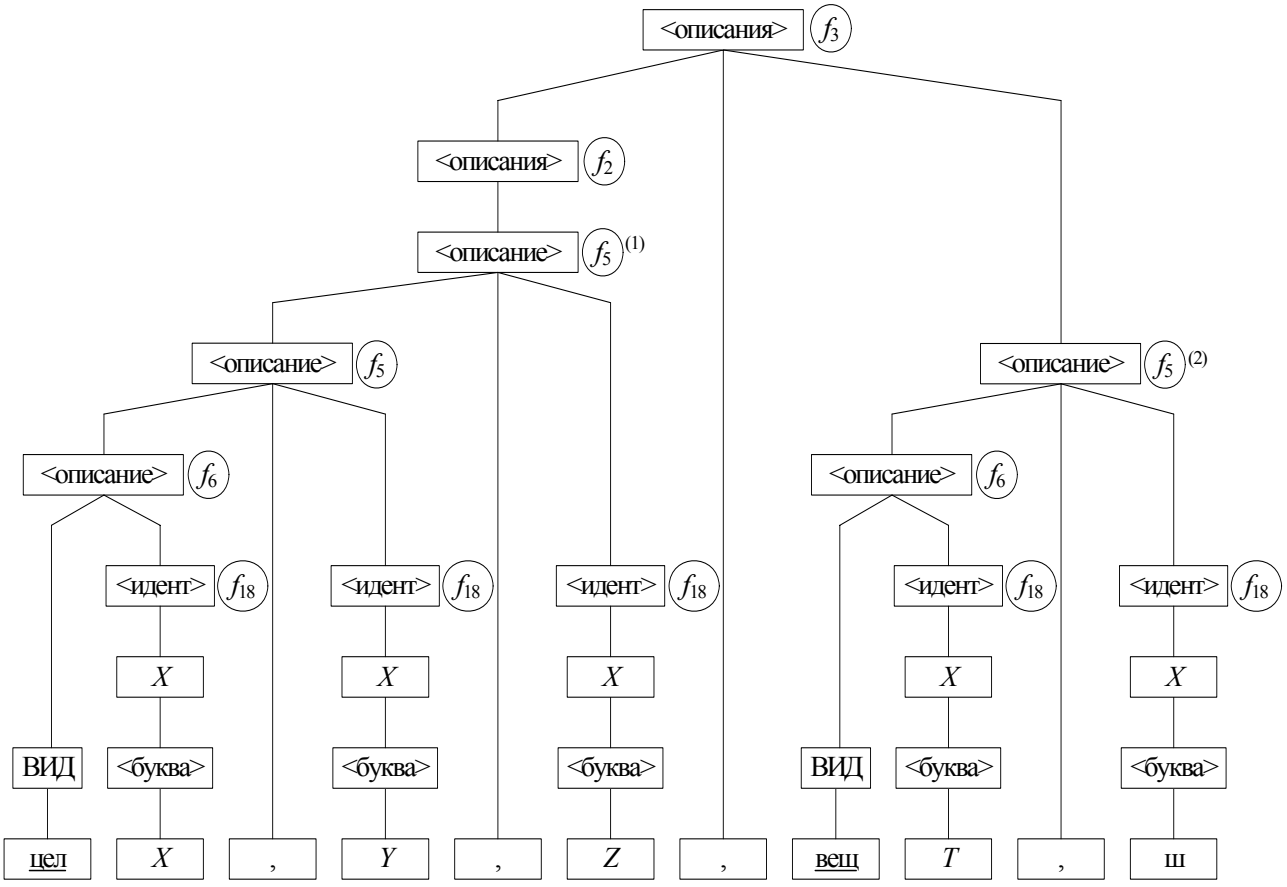


Рис. 4. Дерево синтаксического разбора заданного фрагмента программы

Таким образом, заголовок функции играет большую логическую роль при синтаксическом разборе и управлении вычислением суперпозиции функций. Это связано с тем, что каждый нетерминал в качестве своего значения имеет несколько функций. Выбор конкретной функции нетерминал осуществляет в зависимости от контекста, а инструментом данного выбора являются типы, перечисленные в заголовке. Отсюда следует, что заголовок служит фундаментом для построения контекстно-ориентированного программирования и имеет основное значение при синтаксическом разборе предложений естественного языка.

Рассмотрим тело функции f_6 (см. рис. 4). Внутри скобок находятся два нетерминала: ВИД – от грамматики G_{M1} – и X – от грамматики G_0, которые являются формальными параметрами функции. Фактическими параметрами функции f_6 в начале программного фрагмента (это самая левая часть дерева разбора) являются цел и X, при этом цел сворачивается в ВИД по одному правилу, а A как буква сворачивается сначала в нетерминал X, а потом – в нетерминал <идент> $\{f_{18}\}$. Далее ВИД и <идент> сворачиваются в <описание> $\{f_6\}$. Поскольку в заголовке f_6, как уже было указано, аргументы x и y имеют тип текст, то нетерминалы ВИД и X рассматриваются функцией как текстовые значения цел и A. Функция f_{18}, соответствующая нетерминалу <идент>, игнорируется, так как ее результат имел тип знач, а вовсе не тип текст. Отсюда можно сделать вывод, что заголовок функции управляет процессом вычислений в теле функции, а также процессом вычислений в суперпозиции функций, что еще раз подтверждает значимость заголовка функции.

Результат функции f_6 в терминах формальных параметров имеет вид

имя ВИД X #,

где имя, # – термы; ВИД имеет текстовое значение цел, X – текстовое значение A. Значение функции определяется при помощи операции отождествления. Для вычисления f_6, с одной стороны, имеются формальные параметры ВИД и X, с другой стороны – фактические параметры цел и A (рис. 5). Операция отождествления основывается на сопоставлении формальных и фактических аргументов:

ВИД : = цел и X : = A.

Тогда результат функции f_6 равен

имя цел A #.

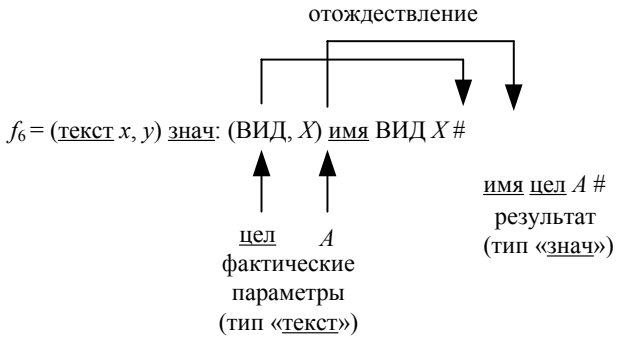


Рис. 5. Вычисление значения функции f_6

Теперь рассмотрим функцию

f_5 = (знач x, текст y) знач: (ВИД X # M, $X^{(1)}$)
ВИД X # M ВИД $X^{(1)}$ #.

Она также имеет два аргумента x и y, но первый аргумент имеет тип знач. Именно поэтому в качестве первого фактического параметра будет выступать результат выполнения функции f_6

имя цел A #.

Отметим, что формальный параметр X имеет в своем составе нетерминал M, который в данном случае равен ε (пустоте). Вторым параметром f_5 является текстовое значение нетерма X, которое равно B. Результатом функции f_5 (рис. 6) будет являться значение

имя ВИД A # имя ВИД B #.

Найденный результат, согласно дереву синтаксического разбора, будет подаваться на вход новой функции $f_5^{(1)}$. В этом случае нетерминал M уже не будет пустым, а равным

имя ВИД B #.

В этом случае результатом функции является значение

имя ВИД A # имя ВИД B # имя ВИД C #.

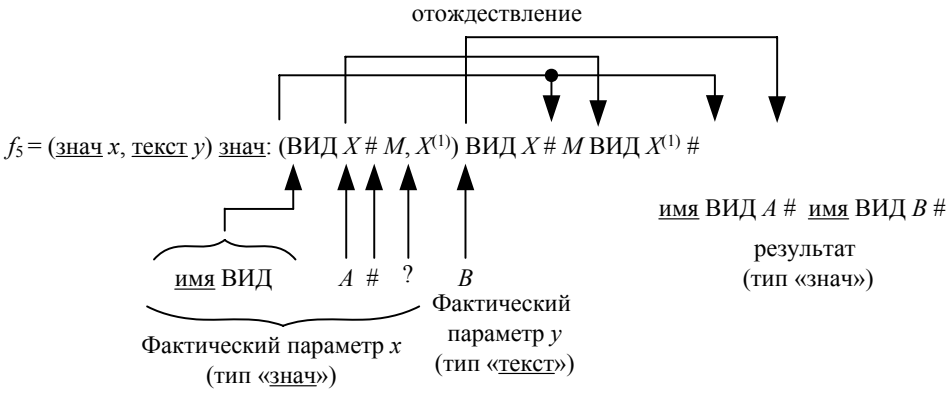


Рис. 6. Вычисление значения функции f_5

Таким образом, фрагмент программы

цел *A*, *B*, *C*

сворачивается в нетерминал <описание>, которому соответствует результат функции $f_5^{(1)}$:

имя ВИД *A* # имя ВИД *B* # имя ВИД *C* #.

Другой фрагмент

вещ *D*, *E*

также сворачивается в <описание> с функцией $f_5^{(2)}$, значение которой, по аналогии с $f_5^{(1)}$, равно

имя ВИД *D* # имя ВИД *E* #.

Далее символ <описание>, находящийся в левой части дерева, сворачивается в символ <описания> посредством функции

f_2 = (конт *t*, знач *x*) конт: (*E*, *M*) f_3(*E*, ε, *M*).

Стоит обратить внимание на то, что результатом данной функции, имеющей два аргумента, является вызов функции f_3 с тремя параметрами:

f_3 = (конт *t*, знач *x*, *y*) конт: ((*M* ВИД *X* # *E*, ε, ВИД$^{(1)}$ *X* # *M*) ⊥ | (*E*, ε, ВИД *X* # M) f_3 (имя ВИД *X* # *E*, ε, *M*) | (E, ε, ε) знач (ε)).

При этом два аргумента вызывающей функции находят свое место в качестве первого и третьего параметров вызванной функции, а в качестве второго параметра выступает ε. Таким образом, f_2 является вспомогательной функцией вызова f_3. Обе функции работают с общей памятью *E*, о чем нам сообщает тип конт.

Функция f_3 является рекурсивной и состоит из трех альтернатив. В случае выполнения первой альтернативы значением функции будет являться ошибка (⊥). Вторая и третья альтернативы образуют рекурсию. Вид рекурсии

(*E*, ε, ВИД *X* # M) f_3(имя ВИД *X* # *E*, ε, *M*)

говорит о том, что каждый новый вызов функции будет изымать значение

имя ВИД *X*

из нетерминала *M* и добавлять его в память *E*, проставляя каждый раз впереди дополнительно терм имя. Это будет происходить до тех пор, пока согласно выражению

(*E*, ε, ε) знач (ε)

значение нетерминала *M* не будет исчерпано. Таким образом, область значения функции f_2 состоит из декартового произведения множества состояний памяти (конт) и множества значений, которые образуются на исходящей ветви нетерминала f_2 в дереве разбора (знач). Результатом на памяти будет являться состояние памяти

E ≡ имя имя цел *C* # имя имя цел *B* # имя имя цел *A* #,

а результатом на исходящей ветви дерева – пустое значение

ε = знач (ε).

Последнее означает, что нетерминал <описание> не производит передачу результата в суперпозицию функций, вместо этого происходит формальная передача параметра ε.

Полученный результат ε и результат функции $f_5^{(2)}$ подаются (теперь уже непосредственно) на вход функции f_3. Возникающая при этом рекурсия будет аналогична рассмотренной ранее. Ошибка ⊥ описываемая альтернативой

(*M* ВИД *X* # *E*, ε, ВИД$^{(1)}$ *X* # *M*),

необходима для того, чтобы исключить добавление уже имеющейся переменной, объявив для нее новый тип.

В результате значение функции $f_5^{(2)}$ пополнит общую память *E* до состояния

E ≡ имя имя вещ *E* # имя имя вещ *D* # имя имя цел *C* # имя имя цел *B* # имя имя цел *A* #.

Таким образом, с помощью составленной функциональной грамматики мы смогли описать процесс выделения в памяти ячеек для переменных разных типов.

Вычисления на уровне типов, выполняемые компилятором, имеют существенное значение для синтеза программ. При этом контексты, в среде которых находятся нетерминалы, могут управлять синтаксическим разбором и процессами вычислений в суперпозиции функций. Данное управление осуществляется через заголовки базисных функций – нетерминалов.

Библиографические ссылки

1. Тузов В. А. Математическая модель языка. Л. : Изд-во Ленингр. ун-та, 1984.

2. Тузов В. А. Языки представления знаний : учеб. пособие. Л.,1990.

V. A. Kravchenko, P. B. Mognonov, D. N. Chimitov

KNOWLEDGE REPRESENTATION IN FUNCTIONAL GRAMMARS

One of the ways of representation of knowledge (theories) by means of functional grammars is considered in the article. The authors make an attempt to give the uniform scheme of the decision of the problem, by its downsinking in functional grammar of the theory.

Keywords: context-oriented programming, object-oriented programming, data structures.