

Логическое программирование

Анализ и понимание текста

Е. А. Черкашин

ИДСТУ им. В. М. Матросова СО РАН

26 ноября 2021, Иркутск

Парадигмы программирования

Парадигма программирования (ПП.) – совокупность идей и понятий, определяющих стиль проектирования компьютерных программ (П.) (подход к программированию); способ концептуализации, определяющий организацию вычислений и структурирование работы, выполняемой компьютером (Wikipedia).

- ❑ **Императивная П.** – последовательность операторов (C, Pascal).
 - ▶ **Низкоуровневая П.** – (1:1) – управляет непосредственно процессором (Assembler).
 - ▶ **Конкатенационная ПП.:** операторы (слова) можно “склеить” в машинный код (Forth).
 - ▶ **Структурное П.:** П. и структуры данных представляются иерархически (Pascal, C, ...).
 - ▶ **Switch-П.** меняет стратегию поведения при возникновении определенных событий (прием).
- ❑ **Функциональная П.** – суперпозиция функций
 $prog(x, y) = f(x, g(y, 5))$ (LISP, ML, Haskell, OCaml,...).
- ❑ **Логическая П.** – набор высказываний, необходимо доказать истинность (Prolog).

Парадигмы программирования (продолжение)

- ❑ **П. Переписывания** – набор правил замены “одной строки” (шаблона) на другую (Рефал).
 - ▶ **Макро-П.** – “раскрытие” структур (прием).
 - ▶ **BDD¹:** “Раскрытие” поддеревьев (TeX, L^ATeX, ...).
- ❑ **Объектно-ориентированная П.** – набор объектов, посылающих друг другу сообщения.
 - ▶ **Компонентно-ориентированная** – система состоит из модулей, интерфейсы которых доступны во время выполнения ... (COM+, CORBA, ZCA).
 - ▶ **Прототипно-ориентированная:** объекты и классы порождаются клонированием (JavaScript, Lua).
 - ▶ **Агентно-ориентированная П.** – набор взаимодействующих *агентов*, поведение которых программируется независимо (прием).
 - ▶ **Аспектно-ориентированные инструменты** разработки позволяют породить объекты ООП “послойно” (прием, AspectJ).
- ❑ **Порождающее П.** – часть программного кода генерируется из абстрактной модели (прием).
- ❑ **Грамотное П.²** – Документация и П. проектируются одновременно (статья в журнале).

¹Binary Decision Diagrams

²Literate Programming © D. Knuth

Логическое программирование

– ПП., основанная на *автоматическом доказательстве теорем*, а также раздел дискретной математики, изучающий принципы логического вывода *суждений* на основе заданных *фактов* и *правил* вывода.

- “%” – комментарий, простирающийся до конца строки,
- Программа – набор фраз (clause):

- ▶ факты

```
works(mario,plumber).  
likes(mary,apple).  
contains(sedan,wheel,4).  
color(apple,red).  
color(orange,orange).
```

- ▶ правила

```
likes(kate,X) :-    % ':-' – Если  
    likes(mary,X), % ',' – И  
    color(X,red).  % X-переменная
```

- ▶ запросы (суждения)

```
?- likes(kate, X).  
X=apple.
```

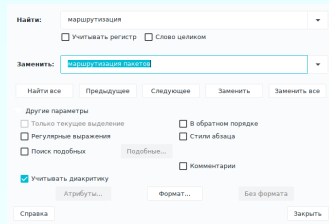
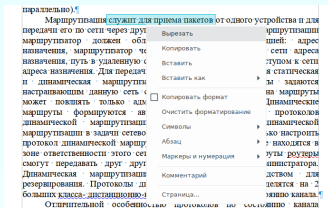
Актуальность понимания естественного языка

Понимание естественного языка (ЕЯ), перевод из одного ЕЯ на другой – **Направление ИИ**. Решаются следующие задачи:

1. Анализ текстов, помещение изъятой информации в базу данных:
 - ▶ изготовление шаблонов документов, отчетов;
 - ▶ синтез структур данных для ИС;
 - ▶ заполнение баз данных ИС и т.п.;
2. Ведение диалога с пользователем:
 - ▶ идентификация моделей и планирование действий (интеллектуальные пользовательские интерфейсы);
 - ▶ приобретение знаний (оболочки экспертных систем);
3. Управление приложением:
 - ▶ запросы на естественном языке к базам данных;
 - ▶ внесение изменений в данные;
 - ▶ командное управление («Проветрить квартиру»).

Графический пользовательский интерфейс

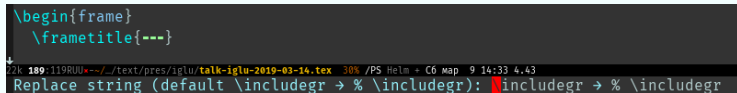
Специализирован на **унарных операциях над отдельными частями** информационного объекта.



В операциях аргументы вводятся в диалоговом окне (для операций с аргументами).

Более «умные» редакторы не используют контекстные операции (EMACS, VI, Visual Studio Code, Sublime, AutoCAD).

Alt-X replace-string, Набирается как «Alt-X repl str»



Семиотика (наука о знаках) делится на три раздела (Моррис)

- ❑ **Семантика** — отношение знака к объекту:
Что значит *знак*?
- ❑ **Синтаксис** — отношение знаков между собой –
как создаются *новые смыслы* (термины, суждения)
комбинированием *знаков*.
- ❑ **Прагматика** — отношение знака к субъекту:
Что *обозначает предложение*, что надо дальше делать? На какой
конкретно вопрос и как надо отвечать?

Язык программирования

Синтаксис на уровне грамматики определяет корректные последовательности символов (операторы, структуры). Но синтаксическая правильность не гарантирует даже осмысленности программы.

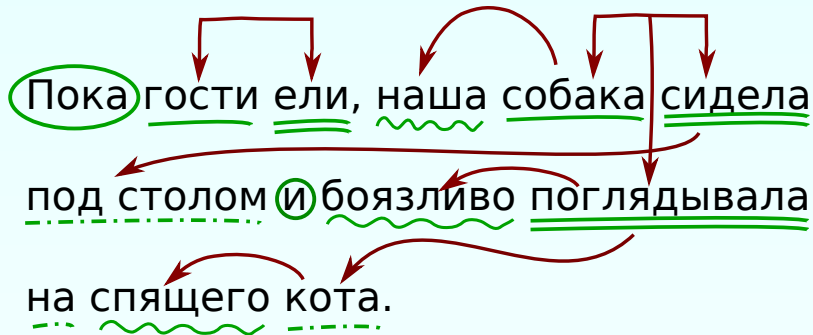
Семантика — это соответствие между синтаксически правильными программами и [вариантами] действий абстрактного исполнителя, то есть это смысл синтаксических конструкций.

Прагматика задает конкретизацию абстрактного вычислителя (конкретный процессор и др. ресурсы) для вычислительной системы. Стандарт языка программирования задаёт поведение вычислителя не полностью, конкретный транслятор языка переводит программу в конкретный машинный код на конкретную программно-аппаратную платформу.

Реализованный язык является прагматическим опосредованием абстрактной модели вычислений и ее реализацией на конкретном компьютере.

Цель программиста — получить нужный ему эффект в результате исполнения программы на конкретном оборудовании: трансляция и исполнение осуществляется на конкретных вычислителях.

Синтаксический разбор предложения



$$G = \langle T, N, \Sigma, R \rangle$$

T – множество терминальных символов (слова, буквы, IF, ELSE),

N – множество нетерминальных символов (обозначения, A, B, <noun>, <verb>), $T \cap N = \emptyset$,

Σ – стартовый символ (<программа>, <предложение>), $\Sigma \in N$,

R – множество правил грамматики

$A \rightarrow B$,

$R \subset ((T \cup N)^* N (T \cup N)^*) \times (T \cup N)^*$.

Язык $L(G) = \{\Omega \in T^* \mid \Sigma \rightarrow^* \Omega\}$.

Вывод $\Sigma \rightarrow^* \Omega$

$\Sigma \rightarrow \Sigma A \quad \Sigma \rightarrow A$

$A \rightarrow b \Sigma e \quad A \rightarrow be$

Пример: $a = ' (, \quad b = ') '$.

Σ	Σ
A	A
$b \Sigma e$	(Σ)
$b \Sigma A e$	(ΣA)
$b A A e$	$(A A)$
$b b e A e$	$(() A)$
$b b e b e e$	$(() ())$

По иерархии Ноама Хомского, грамматики делятся на **четыре** типа, каждый последующий является более ограниченным подмножеством предыдущего (но и легче поддающимся анализу):

- ❑ Тип 0. Неограниченные грамматики — возможны любые правила;
- ❑ Тип 1. Контекстно-зависимые грамматики — левая часть может содержать один нетерминал, окруженный «контекстом»; сам нетерминал заменяется непустой последовательностью символов в правой части;
- ❑ Тип 2. Контекстно-свободные грамматики — левая часть состоит из одного нетерминала;
- ❑ Тип 3. Регулярные грамматики — более простые, распознаются конечными автоматами.

Грамматика языка программирования С

```
<translation-unit> ::= {<external-declaration>}*  
<external-declaration> ::= <function-definition>  
    | <declaration>  
<function-definition> ::= {<declaration-specifier>}* <declarator> {<declaration>}* <compound-statement>  
    | union  
<struct-declaration> ::= {<specifier-qualifier>}* <struct-declarator-list>  
<specifier-qualifier> ::= <type-specifier>  
    | <type-qualifier>  
<struct-declarator-list> ::= <struct-declarator>  
    | <struct-declarator-list> , <struct-declarator>  
<struct-declarator> ::= <declarator>  
    | <declarator> : <constant-expression>  
    | : <constant-expression>  
<selection-statement> ::= if ( <expression> ) <statement>  
    | if ( <expression> ) <statement> else <statement>  
    | switch ( <expression> ) <statement>  
<iteration-statement> ::= while ( <expression> ) <statement>  
    | do <statement> while ( <expression> ) ;  
    | for ( {<expression>}? ; {<expression>}? ; {<expression>}? ) <statement>  
<jump-statement> ::= goto <identifier> ;  
    | continue ;  
    | break ;  
    | return {<expression>}? ;
```

Пример трансляции

```
#include <stdio.h>

typedef unsigned long int ulint;

ulint fact(ulint n) {
    if (n==0) return 1;
    if (n==1) return 1;
    return n*fact(n-1);
}

int main() {
    ulint n = 10;
    printf("Factorial of %lu = %lu.\n",
        n, fact(n));
    return 0;
}
```

```
.file "fact.c"
.text
.globl fact
.type fact, @function

fact:
.LFB11:
    movl    $1, %eax
    cmpq    $1, %rdi
    jbe     .L4

.L3:
    imulq   %rdi, %rax
    subq    $1, %rdi
    cmpq    $1, %rdi
    jne     .L3

.L4:
    ret

.LFE11:
.section .rodata.str1.1,"aMS",@progbits,1
.LC0:
.string "Factorial of %lu = %lu.\n"
.section .text.startup,"ax",@progbits
.globl main
.type main, @function

main:
.LFB12:
;; . . . . .
ret
.cfi_endproc

.LFE12:
.size main, -main
.ident "GCC: (GNU) 8.2.1 20181127"
.section .note.GNU-stack,"",@progbits
```

Синтаксический разбор предложения

Корова трясет хвостом.

A cow shakes the tail.

% [a, cow, shakes, the, tail]

Грамматика:

- ❑ Множество **терминальных** символов – $\{a, b, c, \dots, z\}$. На самом деле, $\Sigma = \{a, cow, shakes, walks, \dots\}$.
- ❑ Множество **нетерминальных** символов – $\langle sentence \rangle$, $\langle noun \rangle$, $\langle verb \rangle$, ...
- ❑ Стартовый символ – $\langle sentence \rangle$.
- ❑ **Правила** упрощенного английского языка:

$$\langle sentence \rangle \rightarrow \langle noungroup \rangle \langle verbgroup \rangle \quad (1)$$

$$\langle noungroup \rangle \rightarrow \langle determinant \rangle \langle noun \rangle \quad (2)$$

$$\langle verbgroup \rangle \rightarrow \langle verb \rangle \langle noungroup \rangle \quad (3)$$

$$\langle noun \rangle \rightarrow cow \mid tail \mid \dots \quad (4)$$

$$\langle verb \rangle \rightarrow walks \mid shakes \mid \dots \quad (5)$$

$$\langle determinant \rangle \rightarrow a \mid the \mid \varepsilon \mid my \mid \dots \quad (6)$$

Пишем программу!

Информационная система GeoBase. База данных

GeoBase – программа, позволяющая делать запросы на ЕЯ к базе данных по географии США, Borland, 1988.

```
state('alabama','al','montgomery',3894e3,51.7e3,22,'birmingham','mobile','montgomery','huntsville').
state('alaska','ak','juneau',401.8e3,591e3,49,'anchorage','fairbanks','juneau','sitka').

city('alabama','al','birmingham',284413).
city('alabama','al','mobile',200452).

border('florida','fl',['georgia','alabama']).

highlow('alabama','al','cheaha mountain',734,'gulf of mexico',0).

mountain('alaska','ak','mckinley',6194).
mountain('alaska','ak','st. elias',5489).

road('66',['district of columbia','virginia']).

lake('huron',59570,['michigan']).
```

The database contains the following information:

Information about states:

- Area of the state in square kilometers
- Population of the state in citizens
- Capital of the state
- Which states border a given state
- Rivers in the state
- Cities in the state
- Highest and lowest point in the state in meters

Information about rivers:

- Length of river in kilometers

Information about cities:

- Population of the city in citizens

Примеры запросов

Some sample queries:

- states
- give me the cities in california.
- what is the biggest city in california ?
- what is the longest river in the usa?
- which rivers are longer than 1 thousand kilometers?
- what is the name of the state with the lowest point?
- which states border alabama?
- which rivers do not run through texas?
- which rivers run through states that border the state with the capital austin?

Смотрим работу GeoBase!

Схемы (спецификации) интерпретации

```
schema('abbreviation','of','state').
schema('state','with','abbreviation').
schema('capital','of','state').
schema('state','with','capital').
schema('population','of','state').

schema('area','of','state').
schema('city','in','state').

schema('length','of','river').
schema('state','with','river').
schema('river','in','state').

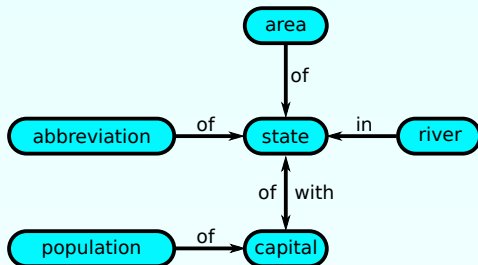
schema('capital','with','population').
schema('point','in','state').

schema('height','of','point').
schema('mountain','in','state').

schema('height','of','mountain').
schema('lake','in','state').

schema('name','of','river').
schema('name','of','capital').

schema('road','in','continent').
```



Основной цикл программы

```
geobase(STR, X, E):-
    STR \= "",
    atom_string(ATOM,STR),
    tokenize_atom(ATOM,LIST),      /* Returns a list of words(symbols) */
    filter(LIST,LIST1),            /* Removes punctuation and words to be ignored*/
    pars(LIST1,E,Q),               /* Parses queries */
    findall(A,eval_interp(Q,A),L),
    unik(L,L1),
    % unit(E,U),
    member(X,L1).
```

Синтаксический анализатор (часть)

```
    pars(LIST,E,Q):-s_attr(LIST,OL,E,Q),check(OL),!.
    pars(LIST,_,_):-error(LIST),fail.
%
/* How big is the biggest city - BIG QUERY */
s_attr([BIG|S1],S2,E1,q_eaq(E1,A,E2,Q)):-
    size(_,BIG),s_minmax(S1,S2,E2,Q),
    size(E2,BIG),entitysize(E2,E1),
    schema(E1,A,E2),!.

s_attr(S1,S2,E,Q):-s_minmax(S1,S2,E,Q).
% . . . . .

/* ... the shortest river in texas - MIN QUERY */
s_assoc1([MIN|S1],S2,E1,A,q_eaq(E1,A,E2,q_min(E2,Q))):-minn(MIN),!,
s_nest(S1,S2,E2,Q),schema(E1,A,E2).

/* ... the longest river in texas - MAX QUERY */
s_assoc1([MAX|S1],S2,E1,A,q_eaq(E1,A,E2,q_max(E2,Q))):-maxx(MAX),!,
s_nest(S1,S2,E2,Q),schema(E1,A,E2).
```

population of Washington

Население штата или города?

?- schema('population','of','city').

?- schema('population','of','state').

Корпус реализован при помощи реструктуризации базы данных.

```
/* Relationships about states */
db(abbreviation,of,state,ABBREVIATION,STATE):- state(STATE,ABBREVIATION,_,_,_,_,_,_,_).
db(state,with,abbreviation,STATE,ABBREVIATION):-state(STATE,ABBREVIATION,_,_,_,_,_,_,_).
db(area,of,state,AREA,STATE):- state(STATE,_,AREA1,_,_,_,_,_,_),str_real(AREA,AREA1).
db(capital,of,state,CAPITAL,STATE):- state(STATE,_,CAPITAL,_,_,_,_,_,_).
db(state,with,capital,STATE,CAPITAL):-state(STATE,_,CAPITAL,_,_,_,_,_,_).
db(population,of,state,POPULATION,STATE):-state(STATE,_,_,POPUL,_,_,_,_,_,_),str_real(POPULATION,POPUL).
db(state,border,state,STATE1,STATE2):-border(STATE2,_,LIST),member(STATE1,LIST).

/* Relationships about rivers */
db(length,of,river,LENGTH,RIVER):- river(RIVER,LENGTH1,_,_,_,_,_,_,_),str_real(LENGTH,LENGTH1).
db(state,with,river,STATE,RIVER):- river(RIVER,_,LIST),member(STATE,LIST).
db(river,in,state,RIVER,STATE):- river(RIVER,_,LIST),member(STATE,LIST).

/* Relationships about points */
db(point,in,state,POINT,STATE):- highlow(STATE,_,POINT,_,_,_,_,_,_).
db(point,in,state,POINT,STATE):- highlow(STATE,_,_,_,POINT,_,_,_,_).
db(state,with,point,STATE,POINT):- highlow(STATE,_,POINT,_,_,_,_,_,_).
db(state,with,point,STATE,POINT):- highlow(STATE,_,_,_,POINT,_,_,_,_).
db(height,of,point,HEIGHT,POINT):- highlow(STATE,_,_,_,POINT,H),str_real(HEIGHT,H),!.
db(height,of,point,HEIGHT,POINT):- highlow(STATE,_,_,_,POINT,H,_,_,_),str_real(HEIGHT,H),!.
% . . . . .
```

Интерпретация запроса ...findall(A,eval_interp(Q,A),L),

...

```
eval_interp(Q, IAns):-
    eval(Q,A),
    e_i(A,IAns).

eval(q_min(ENT,TREE),ANS):-
    findall(X,eval(TREE,X),L),
    entitysize(ENT,ATTR),
    sel_min(ENT,ATTR,99e99,'',ANS,L).

eval(q_max(ENT,TREE),ANS):-
    findall(X,eval(TREE,X),L),
    entitysize(ENT,ATTR),
    sel_max(ENT,ATTR,-1,'',ANS,L).

eval(q_sel(E,gt,ATTR,VAL),ANS):-
    schema(ATTR,ASSOC,E),
    db(ATTR,ASSOC,E,SVAL2,ANS),
    str_real(SVAL2,VAL2),
    VAL2>VAL.

% . . . . .
eval(q_eaq(E1,A,E2,TREE),ANS):-
    eval(TREE,VAL),db(E1,A,E2,ANS,VAL).

eval(q_eaec(E1,A,E2,C),ANS):-db(E1,A,E2,ANS,C).

eval(q_e(E),ANS):-      ent(E,ANS). % EVAL "ATOM"

eval(q_or(TREE,_),ANS):- eval(TREE,ANS).
eval(q_or(_,TREE),ANS):- eval(TREE,ANS).

eval(q_and(T1,T2),ANS):- eval(T1,ANS1),eval(T2,ANS),ANS=ANS1.
```

Заключение

Чтобы компьютер понимал, что хочет пользователь надо реализовать следующие модели:

1. Грамматику языка (синтаксис),
2. Семантику его элементов,
3. Прагматику:
 - ▶ Соответствие предложений языка смыслом и действиям, для этого всегда требуется,
 - ▶ Модель предметной области.

Актуальные (хайповые) проекты по теме – программирование **навыков** голосовых помощников (Алиса, Маруса, Сири, G-Ной, Алекса, Робин, Сяо-Ай, Олег, Дуся, Горыныч, Тайп, Агрегат).

Умный дом.



Спасибо за внимание!