ТЕОРЕМА 1. *Система* L *полна* $\leftrightarrow$ $(\forall x,y)(\exists \tau$ - терм в L$)$ $(\tau(x)=y)$.

Условие полноты в теореме 1 является самым слабым из возможных условий полноты, оно абсолютно необходимо для любых программных средств. Тем самым теорема 1 означает, что РАМ-машины (или косвенная адресация) являются очень сильным программным средством, позволяющим,так сказать, дойти в проблеме полноты до конца, до самого слабого условия полноты.

Множество Q одноместных общерекурсивных функций и предикатов вместе с классом РАМ, рассматриваемым как средство замыкания на Q, образуют функциональную систему (Q,РАМ). Если $[S]_{PAM} = S$, то S называется замкнутым классом.

Опишем все конечно-порожденные замкнутые классы в (Q,РАМ).

ОПРЕДЕЛЕНИЕ 3. Предпорядок $\leq$ назовем эффективно конечным, определяемым набором вычислимых функций $L = \{f_1,\ldots,f_n\}$, если $(\forall x,y)(x \leq y) \leftrightarrow (\exists \tau$ - терм в L$)(\tau(y) = x))$.

ОПРЕДЕЛЕНИЕ 4. Для произвольного множества $M \subseteq N$ множество предикатов R назовем M-ограниченно конечным, если существует конечный набор предикатов $P = \{p_1,\ldots,p_m\}$, такой что $R = \{p|p/M$ является булевой комбинацией предикатов из P$\}$,где p/M обозначает ограничение предиката p на множество M.

ТЕОРЕМА 2. $\mathcal{A}$ - *конечно-порожденный замкнутый класс тогда и только тогда, когда для некоторого эффективно конечного порядка* $\leq$, *для некоторых* M *и* M-*ограниченно конечного множества предикатов* R *имеем:* $F = \{f|(\forall x,y)(x = f(y) \to x \leq y)\}$, $M = \{x|(\forall f \in F)(f(x) = x)\}$ *и* $\mathcal{A} = F \cup R$.

Литература

1. ЕРШОВ А.П., ЛЯПУНОВ А.А. О формализации понятия программы /Кибернетика. - 1967. - №5. - С. 40-57.

2. АХО А., ХОПКРОФТ Дж., УЛЬМАН Дж. Построение и анализ вычислительных алгоритмов. - М.: Мир, 1979.

РЕАЛИЗАЦИЯ ЛОГИЧЕСКОГО ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ ПРОФ

Соловьёв И.П., Санкт-Петербург

В работе демонстрируется применение в логическом программировании языка "Проф" ("ПРОлог" с функциями) [2] - расширения "Пролога" семантически близкими "Рефалу" рекурсивными функциями, предназначенного для создания развитых систем спецификаций с помощью текстовых образцов древовидной формы. Программа на языке "Проф" дополнительно содержит определения функций,функ -

циональные выражения разрешается подставлять в качестве аргу - ментов предикатов. Определение функции - это упорядоченная последовательность альтернативных продукций:

```
имя_функции: {посылка_1 -> [значение_1];
              . . . . .
              посылка_m -> [значение_m]}
```

Общий вид посылки: (p1,...,pn), c1,...,ck, где p1,... ...,pn - образцы аргументов, а c1,...,ck - необязательные дополнительные условия (предикаты), для их доказательства используется бэктрэкинг. В заключение записывается функциональное выражение - значение функции. Для повышения выразительности и емкости записи разрешается использование сокращенных продукций (без заключения), умалчиваемым значением функции считается значение последнего аргумента посылки.

Простейшие образцы - это стандартные образцы, константы и переменные (специфицированные или нет). Произвольные образцы строятся рекурсивно с помощью структурирования (для обозначения используются квадратные скобки и запятые), объединения свойств (x|y), пересечения (x & y), отрицания (~x), именования (имя:образец) и других приемов. Примеры стандартных образцов: s - образец строки, d - десятичной цифры, l - буквы латинского алфавита. Иллюстрация применения стандартных образцов - функция id, распознающая идентификаторы, и функция let_dig,распознающая цепочки букв и цифр. Например, id("a1")="a1", но id("12") вызывает неуспех:

```
let_dig: {""->""; x:(l|d) y -> x let_dig(y)}
id:      {l x -> l let_dig(x)}
```

Новые типизированные образцы-переменные, применяемые аналогично стандартным образцам, образуются из имен определенных в программе функций. Спецификация таких образцов определяется областью допустимых значений аргумента функции, значением же образца, унифицированного некоторым выражением, становится значение функции от соответствующего выражения. Так,используя имя let_dig, для бо́льшей наглядности можно переопределить функцию id: {l let_dig}. Аналогично можно определить функцию,скажем e, распознающую строки, сбалансированные относительно круглых скобок, а затем в программе использовать образец e для выделения таких выражений. Образцы id и e применяются ниже. Также можно строить образцы из функционального выражения с одной неконкретизированной переменной, обозначаемой знаком подчеркивания.Остальные переменные выражения являются параметрами, "подчеркивающими" образец в зависимости от контекста применения.

Рассмотрим задачу проверки истинности формул арифметики с равенством и операциями + и * над ограниченным множеством целых чисел (-2^{15}, 2^{15}). Предположим, что формулы правильно построены, используют лишь связки И, ИЛИ и НЕ, кванторы всеобщности (A) и существования (E) и что все кванторы и скобки для инфиксных операций расставлены. Проверку (успех или неуспех вычисления) осуществляет вызов определяемой ниже функции T с двумя аргументами: первый - параметр образца - список привязанных к данному уровню вложенности переменных (среда), второй - проверяемая строка. Пример вызова T (безуспешного): T([ ],"A(x)E(y)((x=y)∧ ~((x+y)=0))").

Сначала определим вспомогательный образец var, отождествляющийся с подкванторной переменной и принимающий значения вида [имя_переменной,очередное_значение]. Здесь используется предикат "Пролога" gen, генерирующий в процессе бэктрэкинга все целые числа вида 0,±1,±2,...,$\pm 2^{15}$,-1,-2^{15}:

```
var: {"("id"), gen(X) -> [id,X]}
```

Далее определим T и вспомогательные образцы val, op, res:

```
T: {(x, "(" T(x,_)")"); -- здесь _- содержимое скобок
    (x, e1  "∧" e2), !, T(x,e1), T(x,e2);
    (x, e1  "∨" e2), !, (T(x,e1));(T(x,e2));
    (x, "~" ! ~T(x,_))-- здесь _- то,следует за "~"
    (x, "E" ! var T(var x,_)); -- поиск решения
    (x, "A" ! ((var ~T(var,x,_) ! fail)!_));-- перебор
    (x, e1 "=" e2), var(x,e1)=val(x,e2) ->  "OK"}

val: {(v,"("x:val(v,_) op y: val(v,_)")") -> res(x,op,y);
      (e1 [id,x] e2,id) -> x; -- аргумент - переменная
      (_,x)} -- аргумент число
op:   {"+"|"*"}
res: {(x, "+",y) -> x+y; x(x,"* ",y) -> x* y)}.
```

Литература

1. СТЕРЛИНГ Л., ШАПИРО Э. Искусство программирования на языке "Пролог". - М.: Мир, 1990. - 235 с.

2. СОЛОВЬЕВ И.П. Согласование синтаксиса РЕФАЛ-продукций с интерпретатором СВМ //Программное обеспечение новой информационной технологии. - Калинин, 1989. -С. 125-127.