**Шорин В.Д. 71-ПГ 03.06.2020**

**Зачет по дисциплине «Функциональное и логическое программирование»**

**1. Алгоритм вычисления целей в Прологе.**

Каноническая форма цели (вопроса) является конъюнкцией атомарных предикатов, то есть последовательностью подцелей, разделенных запятыми:

Q=Q1, Q2,…, Qn.

Пролог пытается вычислить цель при помощи унификации термов предикатов подцелей с соответствующими элементами в фактах и заголовках правил. Поиск ответа на вопрос напоминает поиск пути в лабиринте: следует поворачивать налево в каждой развилке лабиринта до тех пор, пока не попадете в тупик. В этом случае следует вернуться к последней развилке и повернуть направо, после чего опять следует повернуть налево и так далее. Унификация выполняется слева направо, как и поиск пути в лабиринте.

Некоторые подцели при унификации с некоторыми фактами или правилами могут оказаться неуспешными, поэтому Прологу требуется способ запоминания точек отката, в которых он может продолжить альтернативные поиски решения. Прежде чем реализовать один из возможных путей вычисления подцели, Пролог фактически помещает в программу указатель, который определяет точку, в которую может быть выполнен возврат, если текущая попытка поиска цели будет неудачной.

Если некоторая подцель оказывается неуспешной, то Пролог возвращается влево к ближайшей точке возврата. С этой точки Пролог начинает попытку найти другое решение для неуспешной цели. До тех пор, пока следующая цель на данном уровне не будет успешной, Пролог будет повторять возврат к ближайшей точке возврата. Эти попытки выполняются внутренними подпрограммами унификации и механизмом возврата.

Замечание: единственным способом освободить переменную, связанную в предложении, является откат при поиске с возвратом.

Алгоритм вычисления цели – частный случай правила резолюции применительно к дизъюнктам Хорна. Вопрос Q является правилом без заголовка, аналогом выражения ¬Q. Пусть D – база данных (множество дизъюнктов). На вопрос Q, следует найти такую подстановку σ, для которой множество σ[D∪(¬Q)] невыполнимо. Стратегия выбора очередной пары дизъюнктов для резолюции здесь очень проста: подцели и предложения просматриваются в текстуальном порядке.

**2. Функции высшего порядка. Функциональное значение.**

Над функциями можно выполнять операции, использовать их как операнды, то есть как данные (значения). Функция как значение на Хаскеле записывается в виде:

\x->e

Эта форма записи заимствована из лямбда-исчисления - формальной основы функционального программирования. В этой записи заключены два компонента: x — параметр функции и выражение e, называемое телом функции, в которое может входить переменная x. Например, \x->x+1 — это функция, которая получает значение параметра, связывает его с переменной x и выдаёт результат, больший на единицу полученного значения. Таким образом, функциональное значение – это безымянная функция от одного аргумента.

Функциональное значение называется абстракцией, потому что оно обобщает множество выражений в один класс. Так последняя функция обобщает бесконечное множество выражений: 0+1, 1+1, 2+1 ... Например, если эта функция получит аргумент 5, то получится конкретное выражение 5+1, которое после вычисления даст результат применения функции – 5

Функциональное значение само по себе — это безымянная функция. В программе же гораздо удобнее обращаться к функциям по именам. Поскольку функциональное значение является выражением, его можно использовать в правой части определения переменной и, таким образом, ему можно дать имя. Дадим, к примеру, функции увеличения аргумента на единицу имя inc:

inc = \x->x+1.

Такая форма записи не совсем привычная, обычно мы вслед за именем функции привыкли писать список её аргументов. Поэтому в Хаскеле можно дать и более удобное эквивалентное определение:

inc x = x+1.

В этой записи, как видим, аргумент не заключён в скобки: сказывается стремление разработчиков Хаскеля ограничить число скобок.

Используемые нами до сих пор определения функции — это поименованные функциональные значения. Если мы дадим определение

inc x = x+1.

то это означает, что с именем inc связывается значение \x->x+1.Можно было бы так и определить функцию:

inc =\x->x+1,

просто предыдущая запись удобнее и привычнее.

Необходимо несколько слов сказать о синтаксисе функциональных значений \x->e - это два операнда x и e, связанные составным оператором, объемлющим первый операнд. Для использования оператора в выражениях нужно знать его приоритет и ассоциативность. Вместо этого для оператора построения функционального значения мы будем пользоваться простым правилом: функциональное значение имеет максимально возможное тела. В соответствии с этим правилом определение функции add с со скобками будет таким:

add = (\x->(\y->x+y)) .

Поскольку в Хаскеле подавляющее большинство функций каррированные, предусмотрена сокращённая запись таких функций. Вместо

\x1->\x2-> … \xn->e

можно писать

\x1 x2 … xn->e

то есть функцию add можно определить ещё так:

add = \x y->x+y .

Если не использовать каррирование, то функцию от нескольких аргументов можно представить одноаргументной функцией, принимающей кортеж. Например, функцию сложения двух чисел можно было бы представить так:

add’ (x,y) = x+y .

Отметим разницу в типах этих функций:

add::Num a => a->a->a

add’::Num a => (a,a)->a

Применение к аргументу функции add’ будет очевидно выглядеть так: add’(1,2).