**Лабораторная работа №4**

**Унификация термов**

Все термы в программе описываются следующими структурами данных:

// Общий интерфейс для всех термов

trait Term:

val name: String

def contains(other: Term): Boolean = false

override def toString: String = name

// Интерфейс для атомов (предикатных и функциональных символов, констант и переменных

trait Atom extends Term:

override def contains(other: Term): Boolean = *this* == other

// Предикатный символ (обозначается большими латинскими буквами)

case class PredicateSymbol(override val name: String) extends Atom

// Функциональный символ (обозначается строчными латинскими буквами)

case class FunctionalSymbol(override val name: String) extends Atom

// Константа (обозначается большими латинскими буквами)

case class Constant(override val name: String) extends Atom

// Переменная (обозначается строчными латинскими буквами)

case class Variable(override val name: String) extends Atom

// Выражение – терм, включающий в себя другие термы, т.е. предикат или функция

case class Expression(override val name: String, terms: Term\*) extends Term:

override def contains(other: Term): Boolean = terms.contains(other)

Алгоритм унификации реализуется функцией unify, код которой приведен ниже:

override def unify(e1: Term, e2: Term): Option[Unificator] =

// Шаг 1 – ставим на место первого терма атом, если он есть

val mayBeTerms = (e1, e2) *match*

*case* (\_: Atom, \_) => Some((e1, e2))

*case* (\_, \_: Atom) => Some((e2, e1))

*case* \_ => None

// Шаг 2 – попытка унификации при наличии переменной

val mayBeUnificator = mayBeTerms.flatMap {

*case* (e1, e2) *if* e1 == e2 => Some(EmptyUnificator)

*case* (e1: Variable, e2) *if* !e2.contains(e1) => Some(SeqUnificator(Substitution(e2 -> e1)))

*case* (e1, e2: Variable) => Some(SeqUnificator(Substitution(e1 -> e2)))

*case* \_ => None

}

// Шаг 3 – унификация составных термов

(mayBeTerms, mayBeUnificator) *match*

*case* (Some(\_), unificator @ Some(\_)) => unificator

*case* (Some(\_), None) => None

*case* \_ =>

val (c1, c2) = (e1.asInstanceOf[Expression], e2.asInstanceOf[Expression])

val (f1, t1) = (c1.terms.head, c1.terms.tail)

val (f2, t2) = (c2.terms.head, c2.terms.tail)

unify(f1, f2) *match*

*case* Some(z1) =>

val a1 = z1.apply(t1)

val a2 = z1.apply(t2)

unify(a1, a2) *match*

*case* Some(z2) => Some(z1 compose z2)

*case* \_ => None

*case* \_ => None

На первом шаге проверяем, нет ли в данной паре унифицируемых термов атома (функционального или предикатного символа, переменной или константы). Если есть, то проверяем следующие условия:

* Если оба терма – атомы, и они совпадают, возвращаем пустой унификатор
* Если один из них переменная и не содержится во втором – возвращаем унификатор, состоящий из одной подстановки – замены терма-переменной на второй терм

Если ни один из пары унифицируемых термов не является атомом, унифицируем их, как составные термы - по частям:

* Сначала унифицируем их головы (первые термы)
* Затем рекурсивно унифицируем хвосты (все оставшиеся термы)
* В конце возвращаем результат как композицию унификаторов голов и хвостов составных термов

Алгоритм рекурсивный и позволяет унифицировать пару термов произвольной вложенности.