**Лабораторная работа №1**

**Поиск в ширину в графах И-ИЛИ от данных**

Система продукционных правил загружается из файла, в котором она хранится в нотации XML в следующем формате:

<system>

<rule name=<имя\_правила>> // описание правила

<antecedents> // список антецедентов

<antecedent name=<имя> /> // описание антецедента

. . .

<antecedent name=<имя> />

</antecedents>

<consequent name=<имя> />. // описание консеквента

</rule>

. . .

<rule name=<имя\_правила>>

. . .

</rule>

</system>

Алгоритм основан на поиске в ширину, реализован на языке Scala рекурсивно с использованием неизменяемых структур данных в соответствии с принципами функционального программирования.

Текущее состояние процесса решения инкапсулировано внутри экземпляра класса State, имеющего вид:

final case class State(terms: Seq[Term], mutated: Boolean = false, target: Boolean = false)

где:

* **terms** – текущее состояние рабочей памяти или список открытых вершин (набор термов)
* **mutated** – флаг, указывающий, что состояние было изменено (для очередной вершины из рабочей памяти были построены ее потомки и занесены в список открытых вершин), и требуется продолжать поиск; если false, значит потомков нет, поиск прекращается.
* **target** – флаг, указывающий, что было достигнуто целевое состояние и можно прекратить поиск

Код алгоритма приведен в функции **search**, имеющей вид:

def search(data: SearchData, system: ProductionSystem): Option[Seq[Rule]] =

def mutateState(state: State, applied: Seq[Rule]): (State, Seq[Rule]) =

val applicable = system.rules.filter { rule =>

!applied.contains(rule) && rule.isApplicable(state)

}

(applicable.foldRight(state) { (rule, state) => rule.apply(state) }, applied ++ applicable)

def recur(state: State, applied: Seq[Rule] = Nil): (State, Seq[Rule]) =

// Если цель содержится в списке открытых вершин, прекратить поиск

*if* data.target.terms.forall(state.terms.contains(\_)) *then*

(State(state.terms, target = true), applied)

*else*

// иначе найти все дочерние вершины для первой вершины списка и добавить в список

mutateState(state, applied) *match*

*case* (newState @ State(\_, true, \_), rules) => recur(State(newState.terms), rules)

*case* newState => newState

recur(State(data.initial.terms)) *match*

*case* (state @ State(\_, \_, true), rules) => Some(rules)

*case* \_ => None

На вход алгоритма поступает система продукционных правил **ProductionSystem**, загруженная из файла, и информация для поиска **SearchData**, включающая в себя цель и данные. Цикл организован с помощью локальной функции **recur**, которая принимает на вход начальное состояние и список примененных правил (он же список закрытых вершин) и возвращает новое состояние и новый список примененных правил.

В случае достижения цели в экземпляре State выставляется флаг нахождения решения и поиск заканчивается. В случае, когда на очередном шаге состояние не изменилось, выставляется флаг отсутствия решения и поиск также заканчивается.

В результате возвращается список построенных вершин и список примененных правил, на основе которых строится визуальное представление графа И-ИЛИ для данной задачи.