# ЛАБОРАТОРНАЯ РАБОТА №6

**ТЕМА:** Формирование нечётких множеств в среде CLIPS.

**ЦЕЛЬ:** Научиться использовать возможности и особенности языка CLIPS при решении задач нечёткой логики.

**ПЛАН ЗАНЯТИЯ:**

1. Использование продукционной модели представления знаний в CLIPS.
2. Возможности языка CLIPS при решении задач нечёткой логики.
3. Контрольные вопросы.

## 1. ИСПОЛЬЗОВАНИЕ ПРОДУКЦИОННОЙ МОДЕЛИ ПРЕДСТАВЛЕНИЯ ЗНАНИЙ В CLIPS

В процедурных языках программирования мы оперируем понятиями ***true*** и ***false***, но существует ряд задач, решить которые, используя только чёткую логику нельзя.

Так, например, как определить является ли человек совершеннолетним или несовершеннолетним? Если ему 18 и больше - значит он совершеннолетний, иначе - нет. Здесь возникает вопрос: если до полных 18-и человеку остался один день или один час, то можно ли его уже считать совершеннолетним? Тогда возникает понятие "почти совершеннолетний". Сколько раз мы говорили: "мне почти сколько-то"? Слова "почти", "немного", "несколько", "около того" и т.п. недоступны процедурному языку программирования, который "любит" точность: либо "да", либо "нет". Тогда на первый план выходят интеллектуальные информационные системы (ИИС).

На данный момент не существует полностью универсального программного продукта, который позволял бы решать все виды задач с нечеткими данными. Из-за этого существует множество классификаций ИИС, но следует отметить, что основное ядро ИИС состоит из механизма накопления знаний о предметной области и способа их обработки. При изучении типов ИИС большое внимание уделяется способам представления и обработки знаний: логическим моделям знаний, фреймам, семантическим сетям, системам продукции, нечетким множествам, нейронным сетям, генетическим алгоритмам и др. Тем не менее, несмотря на большой выбор программных средств (MIKE, Prolog, Logo и т.д.),

наиболее распространённым языком представления знаний является CLIPS. Более 80% экспертных систем используют именно его.

Язык CLIPS основан на правилах и позволяет представить знания в виде предложений типа "***Если*** (*условие*), ***то*** (*действие*)". Под "*условием*" понимается некоторое предложение-образец, по которому осуществляется поиск в базе знаний, а под "*действием*" – действия, выполняемые при успешном исходе поиска.

CLIPS использует продукционную модель представления знаний и поэтому содержит три основных элемента:

- ❒ список фактов
- ❒ базу правил
- ❒ блок вывода

На базу фактов и базу правил возлагаются следующие функции:

- ❒ база фактов представляет исходное состояние проблемы;
- ❒ база правил содержит операторы, которые преобразуют состояние проблемы, приводя его к решению.

При формализации знаний существует проблема, затрудняющая использование традиционного математического аппарата. Это проблема описания понятий, оперирующих качественными характеристиками объектов (много, мало, сильный, очень сильный т. п.). Эти характеристики обычно размыты, и не могут быть однозначно интерпретированы, однако содержат важную информацию (например, "одним из возможных признаков гриппа является высокая температура"). Кроме того, в задачах, решаемых интеллектуальными системами, часто приходится пользоваться неточными знаниями, которые не могут быть интерпретированы как полностью истинные или ложные. Существуют знания, достоверность которых выражается некоторой промежуточной цифрой, например ***0,7***. В CLIPS значения таких переменных определяется через так называемые нечеткие множества. Нечеткое множество, в свою очередь, определяется через некоторую базовую шкалу B и функцию принадлежности – $\mu(x), x \varepsilon B,$ принимающую значения в интервале ***[0;1]***. Функция определяет субъективную степень уверенности эксперта в том, что данное конкретное значение переменной соответствует данной ситуации. CLIPS предлагает аппарат для работы с нечёткими множествами, что может быть полезно в области прогнозирования и оценки качества.

# 2. ВОЗМОЖНОСТИ ЯЗЫКА CLIPS ПРИ РЕШЕНИИ ЗАДАЧ НЕЧЕТКОЙ ЛОГИКИ

## Задача 2 "Правдолюбцы и лжецы"

Рассмотрим возможности языка CLIPS. В головоломке ”Правдолюбцы и лжецы” решается одна из задач, возникающих на острове, населенном обитателями двух категорий: одни всегда говорят правду (назовем их правдолюбцами), а другие всегда лгут (их назовем лжецами).

- **P1**. Встречаются два человека, ***A*** и ***B***, один из который правдолюбец, а другой — лжец. ***A*** говорит: "Либо я лжец, либо ***B*** правдолюбец". Кто из этих двоих правдолюбец, а кто лжец?
- **P2**. Встречаются три человека, ***A***, ***B*** и ***C***. ***A*** и говорит: "Все мы лжецы", а ***B*** отвечает: "Только один из нас правдолюбец". Кто из этих троих правдолюбец, а кто лжец?
- **P3**. Встречаются три человека, ***A***, ***B*** и ***C***. Четвертый, проходя мимо, спрашивает ***A***: "Сколько правдолюбцев среди вас?" ***A*** отвечает неопределенно, а ***B*** отвечает: "***A*** сказал, что среди нас есть один правдолюбец". Тут в разговор вступает ***C*** и добавляет: "***B*** врет!" Кем, по-вашему, являются ***B*** и ***C***?

В программе, решающей проблемы подобного класса, будут использованы широкие возможности средств программирования правил в языке CLIPS и продемонстрированы некоторые интересные приемы, например использование контекстов и обратного прослеживания. Мы также рассмотрим, как конструировать и тестировать прототипы, которые приблизительно воспроизводят поведение окончательной программы (технология построения экспертных систем с использованием прототипов — одна из самых распространенных в настоящее время).

## 3.1. Анализ проблемы

Первым этапом любого программного проекта является анализ решаемой проблемы. Предложенные головоломки можно решить, систематически анализируя, что случится, если персонаж, произносящий реплику, является *правдолюбцем*, а что, если он — *лжец*. Обозначим через ***T(A)*** факт, что ***A*** говорит правду и, следовательно, является *правдолюбцем*, а через ***F(A***) — факт, что ***A*** лжет и, следовательно, является *лжецом*.

Рассмотрим сначала головоломку **P1**.

Предположим, что ***A*** говорит правду. Тогда из его реплики следует, что либо ***A*** *лжец*, либо ***B*** *правдолюбец*. Формально это можно представить в следующем виде:

*T(A) => F(A) v T(B)*

Поскольку ***A*** не может быть одновременно и *лжецом* и *правдолюбцем*, то отсюда следует

*T(A)=> T(B).*

Аналогично можно записать и другой вариант. Предположим, что **A** лжет:

*F(A) => -(F(A) v T(B)).*

Упростим это выражение:

*F(A) => -F(A) ^ -T(B) или F(A) => T(A) ^ F(B).*

Сравнивая оба варианта, нетрудно прийти к выводу, что только последний правильный, поскольку в первом варианте мы пришли к выводу, противоречащему условиям (не могут быть правдолюбцами одновременно ***A*** и ***B***).

Таким образом, рассматриваемая проблема относится к типу таких, решение которых находится в результате анализа выводов, следующих из определенных предположений, и поиска в них противоречий (или отсутствия таковых). Мы предполагаем, что определенный персонаж говорит правду, а затем смотрим, можно ли в этом случае так распределить "роли" остальных персонажей, что не будут нарушены условия, сформулированные в репликах.

Однако эти головоломки включают и нечто, выходящее за рамки типовых проблем математической логики, поскольку реплики в них может произносить не один персонаж (как в головоломке **P2**), а на реплику одного персонажа может последовать ответная реплика другого (как в головоломке **P3**). В исходной версии программы, которую мы рассмотрим ниже, это усложнение отсутствует, но в окончательной оно должно быть учтено. Мы покажем, что постепенное усложнение программы довольно хорошо согласуется с использованием правил.

На практике оказывается, что в первой версии программы удобнее всего воспользоваться "вырожденным" вариантом проблемы, т.е. постараться решить ее в тривиальном виде, который, тем не менее, несет в себе многие особенности реального случая. Вот как это выглядит в отношении наших правдолюбцев и лжецов.

- **P0**. ***A*** заявляет: "Я лжец". Кто же в действительности ***A*** — лжец или правдолюбец?

Мы только что фактически процитировали хорошо известный Парадокс Лгуна. Если ***A*** *лжец*, то, значит, он врет, т.е. в действительности он *правдолюбец*. Но тогда мы приходим к противоречию. Если же ***A*** *правдолюбец*, т.е. говорит правду, то в действительности он *лжец*, а это опять противоречие. Таким образом, в этой головоломке не существует непротиворечивого варианта "распределения ролей", т.е. не существует модели в том смысле, который придается ей в математической логике.

Есть много достоинств в выборе для прототипа программы варианта головоломки **P0**.

В головоломке присутствует только один персонаж.

Выражение не содержит логических связок, таких как **И** или **ИЛИ**, или кванторов, вроде *квантора общности* (**все**) и прочих.

Отсутствует ответная реплика.

В то же время существенные черты проблемы в этом варианте присутствуют. Мы по-прежнему должны попытаться отыскать непротиворечивую интерпретацию высказывания **A**, т.е. должны реализовать две задачи, присутствующие в любых вариантах подобной головоломки:

- формировать альтернативные интерпретации высказываниям;
- анализировать наличие противоречий.

## 3.2. Онтологический анализ и представление знаний

Следующий этап — определить, с какими видами данных нам придется иметь дело при решении этого класса головоломок. Какие объекты представляют интерес в мире правдолюбцев и лжецов и какими атрибутами эти объекты характеризуются?

По-видимому, для решения задач этого класса нам придется иметь дело со следующими объектами.

**Персонажи**, произносящие реплики. **Произносимая реплика** характеризует либо самого персонажа, либо прочих персонажей, либо и тех и других. Персонаж может быть либо *правдолюбцем*, либо *лжецом*.

**Утверждение**, содержащееся в реплике. Это утверждение может быть либо целиком лживым (*ложным*), либо абсолютно правдивым (*истинным*).

Немного поразмыслив, мы придем к выводу, что существуют еще и другие объекты, которые необходимо учитывать при решении задач этого класса.

Существует **среда** (мир), которая характеризуется совокупностью наших предположений. Например, существует мир, в котором мы предположили, что ***A*** — *правдолюбец*, а следовательно, высказанное им утверждение (или утверждения) *истинно*. Это предположение влечет за собой разные следствия, которые образуют контекст данного гипотетического мира.

Существует еще нечто, что мы назовем **причинами**, или **причинными связями** (*reasons*), которые связывают высказывания в том или ином гипотетическом мире. Если ***A*** утверждает, что "***B*** — *лжец*", и мы предполагаем, что ***A*** — *правдолюбец*, то это утверждение является причиной (основанием), по которой мы можем утверждать, что в данном гипотетическом мире ***B*** — *лжец*, а следовательно, все утверждения, которые содержатся в репликах, произносимых ***B***., лживы. Отслеживая такие связи между высказываниями, можно восстановить исходное состояние проблемы, если в результате рассуждений мы придем к противоречию.

Естественно, что эти объекты можно представлять в программе по-разному. Онтологический анализ практически никогда не приводит к единственному способу представления. Для CLIPS-программы выберем следующее представление описанных объектов:

```
;;Объект statement (высказывание) связан с определенным
;;персонажем (поле speaker).
;;Высказывание содержит утверждение (поле claim).
;;Высказывание имеет основание - причину (поле reason),
;;по которой ему можно доверять,
;;и тэг (tag) - это может быть произвольный
;;идентификатор,(deftemplate statement)

(field speaker (type SYMBOL))
(multifield claim (type SYMBOL))
(multifield reason (type INTEGER) (default 0))
(field tag (type INTEGER) (default 1))
 )
```

Вместо того чтобы фокусировать внимание на персонаже, во главу угла постаим произносимую им реплику (высказывание), а персонаж отнесем к атрибутам высказывания, т.о. обеспечим возможность

представить определенную головоломку в виде экземпляра шаблона, приведенного ниже.

```
(statement (speaker A) (claim F A))
```

Этот шаблон можно перевести на "человеческий" язык следующим образом: ***"Существует высказывание, сделанное персонажем A, в котором утверждается, что A лжец и тэг этого высказывания по умолчанию получает значение 1".***

Обратите внимание на то, что в поле reason также будет установлено значение по умолчанию (это значение равно ***0***), т.е. мы можем предположить, что никаких предшествующих высказываний, которые могли бы подтвердить данное, в этой задаче не было.

Обратите внимание, что поля *claim* и *reason* имеют квалификатор ***multifield***, поскольку они могут содержать несколько элементов данных.

Однако недостаточно только представить в программе высказывания персонажей — нам понадобится также выявить суть содержащихся в них утверждений. Далее, приняв определенное предположение о правдивости или лживости персонажа, которому принадлежит высказывание, можно построить гипотезу об истинности или лживости этого утверждения. С каждым таким утверждением свяжем уникальный числовой идентификатор.

Утверждение, смысл которого, например, состоит в следующем: "***T A***..." означает, что ***A*** *правдолюбец*, "***F A*** ..." означает, что ***A*** *лжец*. Утверждение может иметь под собой основание (*reason*) - обычно это тэг высказывания (объекта *statement*) или тэг другого утверждения (объекта *claim*). Утверждение также характеризуется признаком scope, который может принимать значение "*истина*" или "*ложь*".

```
(deftemplate claim
(multifield content (type SYMBOL))
(multifield reason (type INTEGER) (default 0))
(field scope (type SYMBOL)) )
```

Например, раскрыв содержимое приведенного выше высказывания в предположении, что ***A*** говорит правду, получим следующее утверждение (объект *claim*):

```
(claim (content F A) (reason 1) (scope truth)).
```

Таким образом, объект *claim* наследует содержимое от объекта *statement*. Последний становится обоснованием (*reason*) данного утверждения. Поле *scope* объекта *claim* принимает значение предположения о правдивости или лживости этого высказывания.

Еще нам потребуется представление в программе того мира (*world*), в котором мы в настоящее время находимся. Объекты *world* порождаются в момент, когда мы формируем определенные предположения. Нужно иметь возможность различать разные множества предположений и идентифицировать их в программе в тот момент, когда процесс размышлений приводит нас к противоречию. Например, противоречие между высказываниями ***T(A***) и ***F(A)*** отсутствует, если они истинны в разных мирах, т.е. при разных предположениях. Если у вас есть на сей счет сомнения, вернитесь вновь к примерам в самом начале раздела.

Миры будем представлять в программе следующим образом:

```
;;Объект world представляет контекст,
;;сформированный определенными предположениями
;;о правдивости или лживости персонажей.
;;Объект имеет уникальный идентификатор в поле tag,
;;а смысл допущения - истинность или лживость -
;;фиксируется в поле scope,
(deftemplate world
(field tag (type INTEGER) (default 1))
(field scope (type SYMBOL) (default truth)) )
```

Обратите внимание на то, что при указанных в шаблоне значениях по умолчанию мы можем начинать каждый процесс вычислений с объекта *world*, имеющего в поле значение ***1***, причем этот "мир" можно заселить высказываниями персонажей, которых мы предположительно считаем правдолюбцами. Таким образом можно инициализировать базу фактов *the-facts* для задачи **P0** следующим образом:

```
;; Утверждение, что A лжец.
(deffacts the-facts
(world)
(statement (speaker A) (claim FA)) )
```

Если этот оператор ***deffacts*** будет включен в тот же файл, что и объявления шаблонов (а также правила, о которых речь пойдет ниже), то после загрузки этого файла в среду CLIPS нам понадобится для запуска программы дать только команду ***reset***.

### 3.3. Разработка правил

В этом разделе мы рассмотрим набор правил, который помогает справиться с вырожденной формулировкой **P0** задачи о лжецах и правдолюбцах. Первые два правила, *unwrap-true* и *unwrap-false*, извлекают

содержимое высказывания в предположении, что персонаж, которому принадлежит высказывание, является соответственно правдолюбцем или лжецом, и на этом основании формируют объект *claim*.

```
;; Извлечение содержимого высказывания,
(defrule unwrap-true
(world (tag ?N) (scope truth))
(statement (speaker ?X) (claim $?Y) (tag ?N))
 =>
(assert (claim (content T ?X) (reason ?N)
(scope truth)))
(assert (claim (content $?Y) (reason ?M)
(scope truth)))
 )
(defrule unwrap-false
(world (tag ?N) (scope falsity))
(statement (speaker ?X) (claim $?Y) (tag ?N))
 =>
(assert (claim (content F ?X) (reason ?N)
(scope falsity)))
(assert (claim (content NOT $?Y) (reason ?N)
(scope falsity))
 )
```

В каждом из приведенных правил первый оператор в условной части делает предположение соответственно о правдивости или лживости персонажа, а второй оператор в заключительной части правила распространяет предположение на формируемые утверждения — объекты *claim*.

Далее нам понадобятся правила, которые введут отрицания в выражения. Поскольку —***T(A)*** эквивалентно ***F(A)***, а — ***F(A)*** эквивалентно ***T(A)***, то правила, выполняющие соответствующие преобразования, написать довольно просто. Анализ результатов применения этих правил значительно упростит выявление противоречий, следующих из определенного предположения.

```
;; Правила отрицания (defrule notl
?F <- (claim (content NOT T ?P))
 =>
(modify ?F (content F ?P))
 )
(defrule not2
```

```
?F <- (claim (content NOT F ?P))
 =>
(modify ?F (content T ?P))
 )
;; Выявление противоречия между предположением о
;; правдивости и следующими из него фактами,
(defrule contra-truth
(declare (salience 10))
?W <- (world (tag ?N) (scope truth))
?S <- (statement (speaker ?Y) (tag ?N))
?P <- (claim (content T ?X) (reason ?N) (scope truth))
?Q <- (claim (content F ?X) (reason ?N) (scope truth))
 =>
(printout t crlf
"Statement is inconsistent if " ?Y " is a knight."
;; "Высказывание противоречиво, если " ?Y " правдолюбец."
t crlf)
(retract ?Q)
(retract ?P)
(modify ?W (scope falsity))
 )
```

Если предположить, что исходное высказывание было правдивым, то в дальнейшем обнаруживается противоречивая пара утверждений, которые затем удаляются из рабочей памяти, а значение "правдивости" предположения в объекте *world* изменяется на *falsity* (лживость). Если же после этого также будет обнаружено противоречие, то мы приходим к выводу о глобальной несовместимости условий задачи, т.е. в данной постановке мы имеем дело с логическим парадоксом.

```
;; Выявление противоречия между предположением о
;; лживости и следующими из него фактами, (defrule contra-
falsity
(declare (salience 10))
?W <- (world (tag ?N) (scope falsity))
?S <- (statement (speaker ?Y) (tag ?N))
?P <- (claim (content F ?X) (reason ?N) (scope falsity))
?Q <- (claim (content T ?X) (reason ?N)
(scope falsity)) => (printout t crlf
"Statement is inconsistent if " ?Y " is a knave. "
;; "Высказывание противоречиво, если " ?Y " лжец." t crlf)
```

```
(modify ?W (scope contra))
```

Правило *sweep* обеспечивает проверку, все ли следствия из неверного предположения удалены из памяти.

```
;; Удалить из базы фактов все утверждения,
;; которые следуют из предположения о правдивости.
(defrule sweep
(declare (salience 20))
(world (tag ?N) (scope falsity))
?F <- (claim (reason ?N) (scope truth)) =>
(retract ?F)
```

Обратите внимание на то, что правила *contra-truth*, *contra-falsity* и *sweep* имеют более высокий приоритет (значение параметра *salience*), чем другие правила. Этим обеспечивается как можно более ранее обнаружение противоречия, а следовательно, и удаление из базы фактов утверждений, сделанных на основе предположения, приведшего к противоречию. Если теперь запустить на выполнение программу, представив ей исходный набор фактов, соответствующих условию задачи **P0**, то программа обнаружит, что оба контекста противоречивы. Другими словами, независимо от того, предполагаем ли мы, что ***A*** говорит правду или лжет, программа обнаружит противоречие в контексте *world*.

## 4. КОНТРОЛЬНЫЕ ВОПРОСЫ:

**1.** Основные элементы языка CLIPS.
**2.** Какие способы представления и обработки знаний Вы знаете?
**3.** Дайте понятие нечеткой логики.
**4.** Как формируются правила в CLIPS?

# ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

## Задание №1

1. С помощью языка CLIPS опишите систему фактов, которая бы оперировала с понятиями птица, животное, летает и т.д. Начальные факты в базе фактов задавать в виде:

   `animal <животное> is <тип>`
2. Создайте правила для определения тех животных которые летают (для простоты будем предполагать что летают птица, за небольшим исключением)
3. Создайте правила для определения съедобности животного.
4. Создайте правила для выбора наиболее легко-съедаемого животного (с точки зрения сложности поимки, летает/не летает и с точки зрения его съедобности)

## Задание №2

1. Создайте программу, которая бы управляла роботом. В задачу робота входит создание пирамиды из квадратных блоков, лежащих на земле. Блоков с одинаковым размером не бывает. Команды роботу выдавать с помощью (printout). Должно быть две команды: get, put.
2. Усовершенствуйте программу для случая, когда на земле могут лежать блоки одинакового размера образом, таким образом чтобы пирамида складывалась из преимущественно разноцветных блоков.
3. Усовершенствуйте программу для случая, когда блоки имеют разную конфигурацию (квадраты, треугольники, ромбы и т.д.) и пусть робот складывает каждый из блоков в соответствующую пирамиду – треугольную, квадратную, ромбовидную и т.д. Причем преимущественно разноцветную. Выбор пирамиды осуществляется по новой команде – move – пока блок находится в манипуляторе робота.
4. Усовершенствуйте программу таким образом, чтобы она складывала все блоки в пирамиды.
5. Добавьте правила управления вторым роботом – погрузчиком. Который бы вызывал машину соответствующей массы и отправлял на ней законченные пирамиды.