Лабораторная работа №2: Построение экспертных систем с использованием неупорядоченных фактов (шаблонов) и различных типов условных элементов в антецедентах правил

Цель работы

Изучение примеров программирования экспертной системы с использованием неупорядоченных фактов (шаблонов) и различных типов условных элементов в антецедентах правил.

Основные теоретические положения

Неупорядоченные факты представляют собой список взаимосвязанных именованных полей, называемых слотами. Наличие имен полей позволяет осуществлять доступ к полям по именам, в отличие от упорядоченных фактов, где поля специфицируются своим местоположением в факте. Существует два типа слотов: одиночные и мультислоты. Одиночный слот (или просто слот) содержит единственное поле, тогда как мультислот может содержать любое число полей.

Для спецификации состава неупорядоченных фактов (содержащихся в них слотов) используются шаблоны, которые задаются конструкцией `deftemplate`. Синтаксис конструкции `deftemplate` определен ниже:

```
(deftemplate <имя шаблона> ["<комментарий>"]
 <определение слота-1>
  ...
 <определение слота-N>)
```

Пример шаблона, содержащего три одиночных слота представлен ниже:

```
(deftemplate object "Шаблон объекта"
  (slot name)
  (slot location)
  (slot weight))
```

Пример конкретного неупорядоченного факта на основе данного шаблона представлен ниже:

```
(object (name table) (location "room") (weight 15))
```

Синтаксис антецедентов правил. Антецедент правила состоит из последовательности условных элементов – УЭ (conditional elements - CEs), которые должны удовлетворяться, чтобы правило было помещено в агенду. В CLIPS используется шесть основных типов условных элементов: УЭ на основе образца, УЭ-проверка, УЭ «И», УЭ «ИЛИ», УЭ «НЕ», УЭ «существует», УЭ «для всех», логические УЭ. Ниже рассмотрены наиболее часто используемые типы условных элементов, необходимые для выполнения данной лабораторной работы.

УЭ на основе образца (УЭ-образец) – основной и чаще всего используемый тип условного элемента. Он состоит из совокупности ограничений на поля, масок полей (wildcards) и переменных, используемых в качестве ограничений для фактов и объектов, которые сопоставляются с образцом условного элемента. УЭ-образец удовлетворяется каждой сущностью, которая удовлетворяет его ограничениям. Ограничение на поле представляет собой в общем случае совокупность ограничений, которые используются для проверки единственного поля или слота факта или объекта. Ограничение может состоять из единственного литерала или из нескольких связанных ограничений. Кроме литеральных ограничений, поддерживает три других типа ограничений: связанные ограничения (connective constraints), предикатные ограничения и ограничения возвращаемым значением.

Литеральное ограничение задает точное значение (константу) целого, вещественного, символьного или строкового типа, которое должно сопоставляться с полем. При работе с объектами литеральное ограничение задает имя экземпляра. УЭ-образец с литеральными ограничениями не содержит полей масок и переменных. Все ограничения литерального образца должны точно совпадать со всеми полями сопоставляемой сущности.

Упорядоченный УЭ-образец, содержащий только литералы, имеет следующий синтаксис:

```
(<константа-1> ... <константа-n>)
```

Пример:

```
(data 1 "two")
```

УЭ-образец на основе шаблона, содержащий только литеральные ограничения, имеет следующий синтаксис:

```
(<имя шаблона>
 (<имя слота-1> <константа-1>)
  ...
 (<имя слота-n> <константа-n>))
```

Пример литерального условного элемента для неупорядоченного факта: `(person (name Bob) (age 20))`.

Маски полей (wildcards) используются в условных элементах для указания несущественности некоторого поля или группы полей, что позволяет игнорировать некоторые поля в процессе сопоставления. Считается, что маска сопоставляется с любым значением. Используются одно- и многоместные маски. Одноместная маска обозначаемая символом «?» и сопоставляется с любым значением, занимающим точно одно поле в соответствующем месте сопоставляемой сущности. Многоместная маска обозначаемая парой символов «$?» и сопоставляется с любыми значениями, занимающими произвольное число полей в

сопоставляемой сущности. Одно- и многоместные маски могут использоваться в одном образце в любых комбинациях. Не допускается использование многоместной маски в одноместном слоте (содержащем единственное поле) неупорядоченных фактов или объектов. Маски могут комбинироваться с литеральными ограничениями в одном УЭ.
Пример. Образец `(data $? YELLOW $?)` будет сопоставляться со всеми упорядоченными фактами, содержащими в любом поле, кроме первого, символьное значение YELLOW. В частности, он будет сопоставляться со следующими фактами:

```
(data YELLOW blue red green)
(data YELLOW red)
(data red YELLOW)
(data YELLOW)
(data YELLOW data YELLOW)
```

Последний факт будет сопоставляться с образцом дважды, т.к. значение YELLOW содержится в нем два раза.

Переменные используются для запоминания значений полей, чтобы их можно было использовать в дальнейшем в других условных элементах антецедента или в операторах консеквента правила. Таким образом, в отличие от масок, переменные позволяют «захватить» значения переменных, содержащихся в сопоставляемой сущности, для дальнейшего использования. Используются одно- и многоместные переменные со следующим синтаксисом:

```
<одноместная переменная> ::= ?<переменная>,
<многоместная переменная> ::= $?< переменная>,
```

где <переменная> является символьным типом, но должна начинаться с буквенного символа. В имени переменой не допускается использование кавычек. При первом появлении переменная работает так же, как в маске, т.е. связывается с любым значением в данном поле(ях). Последующие появления переменной требуют, чтобы поле сопоставлялось со связанным значением переменной. Имена переменных являются локальными в пределах каждого правила.
Пример. Задано правило:

```
(defrule find-data
  (data ?x $?y ?z)
=>
  (printout t "?x = " ?x " : " "?y = " ?y " : " "?z = " ?z crlf))
```

и следующее множество фактов:

```
(data 1 blue)
(data 1 blue red)
(data 1 blue red 6.9)
```

Тогда при срабатывании правила будет выведен следующий результат:

```
?x = 1 : ?y = (blue red) : ?z = 6.9
?x = 1 : ?y = (blue) : ?z = red
?x = 1 : ?y = () : ?z = blue
```

Ограничения со связками позволяют объединить несколько индивидуальных ограничений и переменных с помощью операций & (и), | (или), и ~ (не). Старшинство операций обычное: ~, &, | – по убыванию. Исключением является случай, когда первым ограничением является переменная, после которой следует связка &. В этом случае первая переменная трактуется как отдельное ограничение, которое также должно удовлетворяться. Например, ограничение `?x&red|blue` трактуется как `?x&(red|blue)`, а не как `(?x&red)|blue`. Синтаксис ограничений со связками:

```
<term-1>&<term-2> ... &<term-3>,
<term-1>|<term-2> ... |<term-3>,
~<term>,
```

где `<term>` – одно- или многоместная переменная, константа или ограничение со связками. Примеры УЭ, содержащих ограничения со связками:

```
(data-A ~blue)
(data-A ?x&~green)
(data-B (value ~green|red))
(data-B (value ?x&~red&~green))
```

Предикатное ограничение используется в тех случаях, когда необходимо ограничить поле, основываясь на истинности некоторого булевого выражения. Для этого используется предикатная функция, возвращающая символьное значение FALSE в случае неудачи и другое значение, в случае успеха. Функция вызывается в процессе сопоставления с образцом. Если она возвращает значение FALSE, то ограничение не удовлетворяется, в противном случае – оно удовлетворяется. Предикатное ограничение задается с помощью символа «:», за которым следует вызов предикатной функции.

Предикатное ограничение может использоваться в конъюнкции с ограничением со связками, а также связанной переменной. В последнем случае переменная сначала связывается некоторым значением, а затем к ней применяется предикатное ограничение. В таком варианте предикатные ограничения часто применяются для проверки типов данных. При этом в качестве предикатных функций используются следующие встроенные функции CLIPS:

- `(numberp` <выражение>) – функция возвращает значение TRUE, если <выражение> имеет тип integer или float, в противном случае возвращается символ FALSE;
- `(floatp` <выражение>) – функция возвращает значение TRUE, если <выражение> имеет тип float, в противном случае возвращается символ FALSE;
- `(integerp` <выражение>) – функция возвращает значение TRUE, если <выражение> имеет тип integer, в противном случае возвращается символ FALSE;
- `(lexemep` <выражение>) – функция возвращает значение TRUE, если <выражение> имеет тип symbol или string, в противном случае возвращается символ FALSE;
- `(symbolp` <выражение>) – функция возвращает значение TRUE, если <выражение> имеет тип symbol, в противном случае возвращается символ FALSE;
- `(stringp` <выражение>) – функция возвращает значение TRUE, если <выражение> имеет тип string и символ FALSE в противном случае.

Примеры использования предикатных ограничений:

```
(data ?x&:(numberp ?x)
```

```
(data ?x&~:(symbolp ?x)
(data ?x&:(> ?x ?y)
```

Ограничение возвращаемым значением использует в качестве ограничения поля значение, возвращаемое некоторой функцией, которая вызывается непосредственно из условного элемента. Возвращаемое значение должно быть одного из примитивных типов. Это значение подставляется непосредственно в образец на позицию, из которой была вызвана функция, как если бы оно было литеральным ограничением. При этом функция вычисляется каждый раз, когда проверяется ограничение. Данное ограничение, так же как функция сравнения, использует символ «=».
Пусть, например, в базе данных хранятся факты на основе следующего шаблона:

```
(deftemplate data (slot x) (slot y))
```

Тогда следующий образец будет сопоставляться с фактами, у которых значение второго слота в два раза больше, чем первого:

```
(data (x ?x) (y =(* 2 ?x)))
```

УЭ-проверка используется для оценки выражений в процессе сопоставления с образцом и имеет следующий синтаксис:

```
(test <вызов функции>)
```

УЭ-проверка удовлетворяется, если вызываемая в нем функция возвращает значение, отличное от FALSE, и не удовлетворяется в противном случае. Использование данного типа УЭ позволяет, в частности, проверять любые соотношения между значениями различных полей (слотов) фактов.
Пример. В следующем правиле проверяется, что абсолютная разность между значениями фактов `data` и `value` не меньше трех:

```
(defrule example-1
  (data ?x)
  (value ?y)
  (test (>= (abs (- ?y ?x)) 3))
=> ...)
```

Постановка задачи

Необходимо сформировать модельную базу данных с помощью команд `deftemplate` и `deffacts`. Далее используя только литеральные ограничения, составить правила для нахождения в БД фактов, удовлетворяющих заданным условиям. Применить УЭ-проверки в составленных правилах. Изменить сформированные правила путем добавления в антецеденты условных элементов с ограничением по возвращаемому значению.

Типы и допустимые значения для слотов

Имя слота	Тип значения	Допустимые значения
name	symbol	любые имена
age	integer	17 – 22
year	integer	2 – 5
spec	string	«hard», «soft», «ai»
aver_mark	float	в интервале [3; 5]

Порядок выполнения работы

1. Сформировать базу данных, содержащую не менее десяти неупорядоченных фактов на основе следующего шаблона:

```
(deftemplate student
  (slot name)         ; имя студента
  (slot age)          ; возраст
  (slot year)         ; год обучения (курс)
  (slot spec)         ; специализация
  (slot aver_mark))   ; средний балл
```

2. Сохранить конструкции `deftemplate` и `deffacts` в файле.
3. Составить в соответствии с вариантом задания правила, реализующие описанные ниже функции, с использованием заданных типов условных элементов. Правила, соответствующие различным пунктам задания, следует сохранять в разных файлах, чтобы демонстрировать их работу преподавателю по отдельности.
    1. Используя только литеральные ограничения, составить правила для нахождения в БД фактов, удовлетворяющих заданным в табл. 1 условиям, и выдачи соответствующих сообщений.
    2. Изменить сформированные в п. *a.* правила путем добавления в антецедент новых условий и изменения выводимых сообщений в соответствии с табл. 2. При реализации новых УЭ использовать УЭ-проверки `(test-CE)`.
    3. Изменить сформированные в п. *b.* правила путем добавления в антецеденты предикатных условных элементов для проверки типов значений слотов в соответствии с табл. 3.
    4. Изменить сформированные в п. *c.* правила путем добавления в антецеденты условных элементов с ограничением по возвращаемому значению. Условия ограничения для различных вариантов приведены в табл. 4.

Возраст поступления в университет определяется по формуле: $[age] - [year]$. Возраст окончания: $[age] + (5 - [year])$.

Варианты заданий

Таблица 1

№ варианта	Условие в антецеденте правила	Сообщение, выводимое в консеквенте правила
1	Студент 2-го курса	Студент 2-го курса <name> учится по специализации <spec>.
2	Возраст студента 20 лет	Студент <name> учится на <year> курсе.
3	Специализация студента «ai»	Студент <name> учится по специализации “ai” на <year> курсе.
4	Средний балл студента 4.0	Студент <name> учится на <year> курсе со средним баллом 4.0.
5	Студент 5-го курса специализации «hard»	Студент <name> учится на 5-м курсе, возраст <age>.
6	Возраст студента 18 лет	Студент <name> имеет средний балл <aver_mark>.
7	Специализация студента «soft»	Студент <name> учится по специализации “soft” на <year> курсе.
8	Средний балл студента 4.5	Студенту <name> <age> лет, он учится на <year> курсе.

Таблица 2

№ варианта	Условие в антецеденте правила	Сообщение, выводимое в консеквенте правила
1	Студент 2-го курса, средний балл не ниже 4.5	Студент <name> имеет средний балл <aver_mark>.
2	Возраст студента 20 лет, специализации «hard» или «soft»	Студент <name> учится по специализации <spec>.
3	Специализация студента «ai», возраст – не менее 20 лет	Возраст студента <name> <age> лет.
4	Средний балл студента 4.0, курс второй или четвертый	Студент <name> учится на <year> курсе со средним баллом 4.0.
5	Студент 5-го курса специализации «hard», старше 19 лет	Студент <name> учится на 5-м курсе, возраст <age>.
6	Возраст студента 18 лет, средний балл – выше 4.0	Студент <name> имеет средний балл <aver_mark>.
7	Специализация студента «soft», не младше 4-го курса	Студент <name> учится по специализации “soft” на <year> курсе.
8	Средний балл студента 4.5, специализация – не «hard»	Студенту <name> <age> лет, он учится по специализации <spec>.

Таблица 3

№ варианта	Проверяемые значения	Сообщение, выводимое в консеквенте правила
1	<year>, <aver_mark>	Студент <name>: тип значения в слоте <year> – integer, а в слоте <aver_mark> – float.
2	<age>, <year>	Студент <name>: типы значений в слотах <age> и <year> – integer.
3	<spec>, <aver_mark>	Студент <name>: тип значения в слоте <spec> – string, а в слоте <aver_mark> – float.
4	<age>, <aver_mark>	Студент <name>: тип значения в слоте <age> – integer, а в слоте <aver_mark> – float.

№ варианта	Проверяемые значения	Сообщение, выводимое в консеквенте правила
5	<year>, <spec>	Студент <name>: тип значения в слоте <year> – integer, а в слоте <spec> – string.
6	<age>, <spec>	Студент <name>: тип значения в слоте <age> – integer, а в слоте <spec> – string.
7	<name>, <aver_mark>	Студент <name>: тип значения в слоте <name> – symbol, а в слоте <aver_mark> – float.
8	<spec>, <name>	Студент <name>: тип значения в слоте <spec> – string, а в слоте <name> – symbol.

Таблица 4

№ варианта	Условный элемент с ограничением по возвращаемому значению	Сообщение, выводимое в консеквенте правила
1	Оканчивает университет в возрасте не старше 24 лет	Студент <name> оканчивает университет в возрасте не старше 24 лет.
2	Поступил в университет в возрасте 17 лет	Студент <name> поступил в университет в возрасте 17 лет.
3	Оканчивает университет в возрасте не младше 25 лет	Студент <name> оканчивает университет в возрасте не младше 25 лет.
4	Поступил в университет в возрасте не старше 20 лет	Студент <name> поступил в университет в возрасте не старше 20 лет.
5	Оканчивает университет в возрасте 22 лет	Студент <name> оканчивает университет в возрасте 22 лет.
6	Поступил в университет в возрасте младше 19 лет	Студент <name> поступил в университет в возрасте младше 19 лет.
7	Оканчивает университет в возрасте старше 22 лет	Студент <name> оканчивает университет в возрасте старше 22 лет.
8	Поступил в университет в возрасте 18 лет	Студент <name> поступил в университет в возрасте 18 лет.

Содержание отчёта

- Цель работы.
- Краткое изложение основных теоретических понятий.
- Постановка задачи с кратким описанием порядка выполнения работы.
- База данных, оформленная в виде таблицы.
- Результаты работы по каждому заданию с объяснением результатов.
- Общий вывод по проделанной работе.
- Код программы.

Пример выполнения задания

lab2_1.CLP

```
(deftemplate student "Атрибуты студента"
```

```
  (slot name)        ; имя студента
  (slot age)         ; возраст
  (slot year)        ; год обучения (курс)
  (slot spec)        ; специализация
  (slot aver_mark))  ; средний балл

(deffacts students "База данных студентов"
  (student (name John) (age 20) (year 3) (spec "hard") (aver_mark 4.5))
  (student (name Jane) (age 18) (year 2) (spec "hard") (aver_mark 5.0))
  (student (name Jude) (age 22) (year 4) (spec "soft") (aver_mark 3.8))
  (student (name Toma) (age 21) (year 3) (spec "ai")   (aver_mark 3.5))
  (student (name Joan) (age 19) (year 4) (spec "hard") (aver_mark 4.5))
  (student (name Mark) (age 23) (year 5) (spec "soft") (aver_mark 3.0))
  (student (name Fred) (age 20) (year 2) (spec "ai")   (aver_mark 4.0))
  (student (name Eric) (age 24) (year 6) (spec "hard") (aver_mark 4.5))
  (student (name Mary) (age 20) (year 3) (spec "ai")   (aver_mark 3.6))
  (student (name Beth) (age 24) (year 5) (spec "soft") (aver_mark 3.1))
)

(defrule find-student "Задание 1"
  ;...
)
```