Модели представления знаний в Интеллектуальных Системах.

Лекция 11 (Часть 1).

Представление знаний фреймами.

Специальности : 230105, 010501

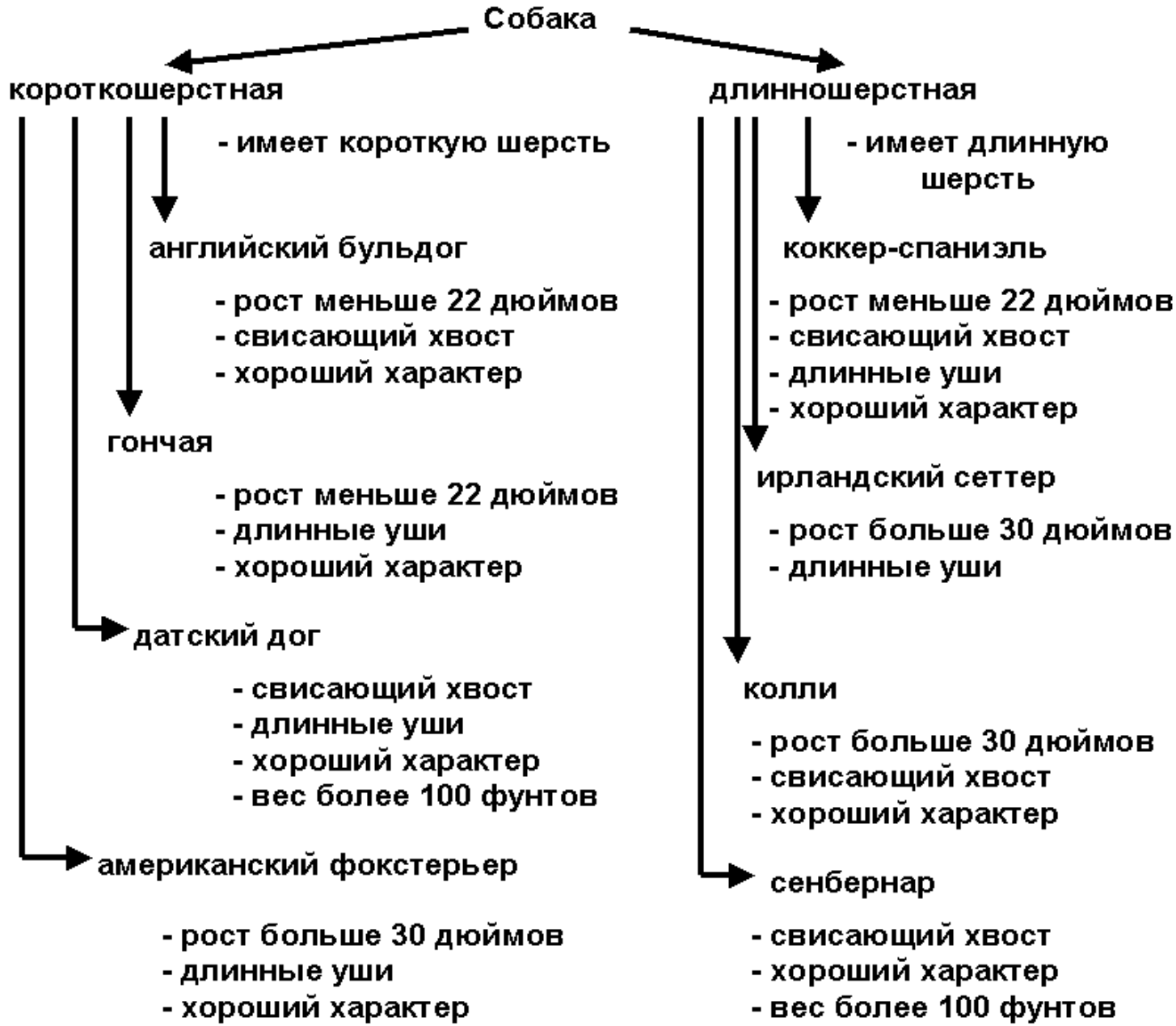
Определение.

Понятие фрейма введено М.Минским [1].

Под фреймом понимается однажды определенная структура данных, которую можно менять лишь в деталях. В основе данной модели представления знаний лежит свойство концептуальных объектов иметь аналогии, которые позволяют строить иерархические структуры отношений типа “абстрактное-конкретное”. Фреймы используются для представления данных и знаний в тех случаях, когда между фрагментами данных либо знаний наблюдается четкая иерархическая зависимость.

Примерами могут служить классификации растений, животных, неисправности радиоаппаратуры, заболеваний человека в медицинской диагностике и многое другое.

Пример : классификация пород собак.



Характеристики фрейма.

- **Имя фрейма – символ. Имя фрейма должно быть уникальным в данной фреймовой системе.**
- **Положение в иерархической структуре. Задается указателями на родительский фрейм и список дочерних фреймов.**
- **Информация, относящаяся к фрейму. Содержится в слотах. Каждый слот может представляться как атомом, так и списочной структурой, первый элемент которой всегда выполняет функцию ключа (соответствует имени слота).**
- **Присоединенные процедуры – служебные программы, являющиеся значениями слотов и запускаемые по сообщениям из других фреймов (аналоги методов в ООП).**

Язык фреймового типа отличается от объектно-ориентированного языка объединением процедурных и декларативных знаний, а также отсутствием специального механизма управления выводом. Достоинство : высокая универсальность.

Описание положения фрейма в структуре (muLISP).

Для описания фреймовых структур на Лиспе можно использовать свойства символов, либо ассоциативные списки.

```
; Функция описывает положение фрейма в структуре
(defun put_frm (frame father slots_info children)
; frame - имя фрейма,
; father - имя родительского фрейма,
; slots_info - информация (слоты),
; children - список дочерних фреймов.
 (put frame 'frm_name frame)
 (put frame 'father father)
 (put frame 'info slots_info)
 (put frame 'children_list children)
 (put father 'children_list (put_obj_to_list frame
                                (get father 'children_list)))
 ((not (null children))
  (put_frm_chlist_processing frame children)))

; Функция включения в список объекта,
; если он там отсутствует
(defun put_obj_to_list (obj lst)
 ((not (member obj lst))(cons obj lst)) lst)

; Обработка списка дочерних фреймов
(defun put_frm_chlist_processing (frame children)
 (put (car children) 'frm_name (car children))
 (put (car children) 'father frame)
 (put (car children) 'info nil)
 (put (car children) 'children_list nil)
 ((not (null (cdr children)))
  (put_frm_chlist_processing frame (cdr children))))

; Определение принадлежности списку
(defun member (obj lst)
  ((null lst) nil)
  ((equal obj (car lst)) T)
  (member obj (cdr lst)))
```

Добавление фрейма в существующую иерархию (muLISP).

```
; Функция добавления фрейма в существующую иерархию
; Аргументы : hierarchy - фреймовая структура,
;                  new_frame - имя добавляемого фрейма,
;                  slots_info - информационное наполнение
                                  (слоты) добавляемого фрейма,
;   childs_of_new_frame - список дочерних фреймов
                                  добавляемого фрейма,
;                  father - имя родительского фрейма.
(defun add_frm (hierarchy new_frame
                  slots_info childs_of_new_frame father)
 ((and (not (equal (get hierarchy frm_name) father))
       (null (get hierarchy children_list))) nil)
 ((equal (get hierarchy frm_name) father)
    (put_frm new_frame father
               slots_info childs_of_new_frame))
 (add_frm_chlist_processing (get hierarchy children_list)
                  new_frame slots_info
                  childs_of_new_frame father))
(defun add_frm_chlist_processing (children new_frame
                  slots_info childs_of_new_frame father)
 ((null children) nil)
 ((not (add_frm (car children) new_frame slots_info
                childs_of_new_frame father))
  (add_frm_chlist_processing (cdr children) new_frame
                  slots_info childs_of_new_frame father))
 (add_frm (car children) new_frame slots_info
                childs_of_new_frame father))
```

```
; add_frm_chlist_processing есть функция
; обработки списка дочерних фреймов
; при добавлении нового фрейма
; в существующую иерархию
```

Описание положения фрейма в структуре (newLISP-tk).

```
; Описание положения фрейма в структуре
(define (put_frm frame father slots_info children)
  (begin
   (set frame (list (list 'frm_name frame)
                    (list 'father father)
                    (list 'info slots_info)
                    (list 'children_list children)))
   (add_to_childrens_for_father frame father_name)
   (put_frm_chlist_processing frame children)))
```

```
(define (add_to_childrens_for_father frame father)
  (cond
   ((null? father) true)
   (true
    (cond
     ((null? (assoc 'children_list father))
      (set father (cons (list 'children_list frame) father)))
     (true (replace-assoc 'children_list father
           (put_obj_to_list frame
             (rest (assoc 'children_list father))
           )))))))
```

```
; Функция включения в список объекта,
; если он там отсутствует
(define (put_obj_to_list obj lst)
  (cond
   ((not (member obj lst)) (cons obj lst))
   (true lst)))
```

```
; Обработка списка дочерних фреймов
(define (put_frm_chlist_processing frame children)
  (cond
   ((null? children) true)
   (true
    (begin
     (set (first children)  (list (list 'frm_name (first children))
                                  (list 'father frame)
                                  (list 'info '())
                                  (list 'children_list '())
                          )
     )
     (put_frm_chlist_processing frame (rest children))))))
```

Примечание. Фреймовые структуры в newLISP-tk реализуются на основе ассоциативных списков.

Управление выводом (muLISP).

```
; Функция сравнения
(defun what_is_it (hierarchy info_descr)
; Функция поиска объекта с заданными свойствами
; по фреймовой структуре. Аргументы :
; hierarchy - фреймовая структура,
; info_descr - информационное описание объекта в виде
;          перечня свойств.
; Условие завершения поиска :
; объект с искомыми свойствами найден
 ((and (null info_descr)
       (null (get hierarchy children_list)))
  (get hierarchy frm_name))
 ((equal (get hierarchy info) info_descr)
  (get hierarchy frm_name))
 ((not (null (compare_slots_info (get hierarchy info) info_descr)))
  (search_childs (get hierarchy children_list)
                 (compare_slots_info (get hierarchy info) info_descr))))

(defun search_childs (frame_list info_descr)
; Просмотр информации дочерних фреймов
((null frame_list) nil)
 ((equal (what_is_it (car frame_list) info_descr) nil)
  (search_childs (cdr frame_list) info_descr))
 (what_is_it (car frame_list) info_descr))

; Функция определяет, является ли список info_frm
; подсписком списка info_search
; (вхождение начинается с головы)
(defun compare_slots_info (info_frm info_search)
; info_frm - списочное описание информационного
;          наполнения слотов анализируемого фрейма
; info_search – список-перечень свойств искомого объекта
 ((null info_frm) info_search)
 ((equal (car info_frm)(car info_search))
  (compare_slots_info (cdr info_frm)(cdr info_search))) nil)
```

Управление выводом (newLISP-tk).

```
; Функция поиска объекта с заданными свойствами
; по фреймовой структуре.
(define (what_is_it hierarchy info_descr)
 (cond
   ((= (first (rest (assoc 'info hierarchy))) info_descr)
    (first (rest (assoc 'frm_name hierarchy))))
   ((not (null?
            (compare_slots_info
              (first (rest (assoc 'info hierarchy)))
              info_descr
            )
    ))
    (search_childs (first (rest (assoc 'children_list hierarchy)))
                   (compare_slots_info
                     (first (rest (assoc 'info hierarchy)))
                     info_descr
                   )))))
(define (search_childs frame_list info_descr)
 (cond
   ((null? frame_list) nil)
   ((null? (what_is_it (eval (first frame_list)) info_descr))
    (search_childs (rest frame_list) info_descr))
   (true (what_is_it (eval (first frame_list)) info_descr))))
; Функция определяет, является ли множество info_frm
; подмножеством множества info_search, и возвращает дополнение
; до разности
(define (compare_slots_info info_frm info_search)
 (cond
   ((null? info_frm) info_search)
   ((= (first info_frm)(first info_search))
    (compare_slots_info (rest info_frm)(rest info_search)))
   (true nil)))
```

Удаление фрейма из структуры (muLISP).

Удаление фрейма из структуры предполагает :

- Удаление имени фрейма из списка дочерних фреймов предка;
- Удаление всех свойств фрейма;
- Удаление всех дочерних фреймов удаляемого фрейма.

```
; Функция удаления фрейма
(defun delete_frm (frame)
 (put (get frame father) 'children_list
       (remove frame (get (get frame father)
        children_list)))
 (remprop frame frm_name)
 (remprop frame father)
 (remprop frame info)
 (delete_childs (get frame children_list))
 (remprop frame children_list))
```

Удаление дочерних фреймов удаляемого фрейма производится в цикле, в каждом проходе которого вызовом функции delete_frm производится удаление очередного фрейма из списка дочерних фреймов frame_list.

```
; Удаление дочерних фреймов
(defun delete_childs (frame_list)
 (loop
  ((null frame_list) nil)
  (delete_frm (pop frame_list))))
```

Для удаления фрейма из списка дочерних фреймов предка здесь используется функция remove, которая удаляет все вхождения объекта-первого аргумента в список-второй аргумент.

Для удаления свойств фрейма используется встроенная функция remprop.

Удаление фрейма из структуры (newLISP-tk).

```
; Функция удаления фрейма
(define (delete_frm frame)
 (begin
   (delete_frm_make (eval frame))
   (delete frame)))

(define (delete_frm_make frame)
 (begin
   (replace-assoc 'children_list
              (eval (first (rest (assoc 'father frame))))
              (delete_from_list (last (assoc 'frm_name frame))
                 (first (rest (assoc 'children_list (eval (first (rest (assoc 'father frame))))))))
   )
   (replace-assoc 'frm_name frame)
   (replace-assoc 'father frame)
   (replace-assoc 'info frame)
   (delete_childs (first (rest (assoc 'children_list frame))))
   (replace-assoc 'children_list frame)))

; Удаление дочерних фреймов
(define (delete_childs frame_list)
 (cond
   ((null? frame_list) true)
   (true (begin
           (delete_frm_make (eval (first frame_list)))
           (delete_childs (rest frame_list))
         ))))
```

```
; Вспомогательная функция удаления объекта из списка

(define (delete_from_list obj lst)
 (cond
   ((null? lst) '())
   ((= (first lst) obj)(delete_from_list obj (rest lst)))
   (true (cons (first lst)
               (delete_from_list obj (rest lst))))))
```

Тестовый пример - построение фреймовой структуры (muLISP).

Следующей последовательностью описанных нами функций в памяти строится фреймовая структура, соответствующая приведенной на плакате 3 классификации пород собак.

```
(put_frm 'dog nil '(it_is_a_dog) '(short-haired long-haired))
(put_frm 'short-haired 'dog nil nil)
(add_frm dog 'short-haired '(short-haired) '(English_bulldog
                                             Beagle
                                             Great_Dane
                                             American_Foxhound) dog)
(add_frm dog 'long-haired '(long-haired) '(Cocker_Spaniel
                                           Irish_Setter
                                           Collie
                                           St_Bernard) dog)
```

Тестовый пример - построение фреймовой структуры (продолжение).

```
; Для короткошерстных пород
(add_frm dog 'English_bulldog
                    '(height_under_22_inches
                      low-set_tail
                      good_natural_personality)
      nil short-haired)
(add_frm dog 'Beagle
                    '(height_under_22_inches
                      longer_ears
                      good_natural_personality)
      nil short-haired)
(add_frm dog 'Great_Dane
                    '(low-set_tail
                      good_natural_personality
                      weight_over_100_lb)
      nil short-haired)
(add_frm dog 'American_Foxhound
                    '(height_under_30_inches
                      longer_ears
                      good_natural_personality)
      nil short-haired)
```

```
; Для длинношерстных пород
(add_frm dog 'Cocker_Spaniel
                    '(height_under_22_inches
                      low-set_tail
                      longer_ears
                      good_natural_personality)
      nil long-haired)
(add_frm dog 'Irish_Setter
                    '(height_under_30_inches
                      longer_ears)
      nil long-haired)
(add_frm dog 'Collie '(height_under_30_inches
                      low-set_tail
                      good_natural_personality)
      nil long-haired)
(add_frm dog 'St_Bernard '(low-set_tail
                      good_natural_personality
                      weight_over_100_lb)
      nil long-haired)
```

Тестовый пример - построение фреймовой структуры (newLISP-tk).

```
(define (frame_struct_example_build)
 (begin
  (put_frm 'dog nil '(it_is_a_dog) '(short-haired long-haired))
  (put_frm 'short-haired 'dog '(short-haired)
           '(English_bulldog Beagle Great_Dane American_Foxhound))
  (put_frm 'long-haired 'dog '(long-haired)
           '(Cocker_Spaniel Irish_Setter Collie St_Bernard))
  (put_frm 'English_bulldog 'short-haired
           '(height_under_22_inches low-set_tail good_natural_personality) '())
  (put_frm 'Beagle 'short-haired
           '(height_under_22_inches longer_ears good_natural_personality) '())
  (put_frm 'Great_Dane 'short-haired
           '(low-set_tail good_natural_personality weight_over_100_lb) '())
  (put_frm 'American_Foxhound 'short-haired
           '(height_under_30_inches longer_ears good_natural_personality) '())
  (put_frm 'Cocker_Spaniel 'long-haired
           '(height_under_22_inches low-set_tail longer_ears good_natural_personality) '())
  (put_frm 'Irish_Setter 'long-haired '(height_under_30_inches longer_ears) '())
  (put_frm 'Collie 'long-haired
           '(height_under_30_inches low-set_tail good_natural_personality) '())
  (put_frm 'St_Bernard 'long-haired
           '(low-set_tail good_natural_personality weight_over_100_lb) '())))
```

Тестовый пример - вывод экспертного заключения.

В предложенной здесь реализации вывода экспертного заключения перечень свойств искомого объекта соответствует утвердительным ответам на вопросы экспертной системы, касающиеся наличия у объекта свойств, представленных в слотах фреймовой структуры.

```
(what_is_it dog '(it_is_a_dog
                  long-haired
                  height_under_30_inches
                  low-set_tail
                  good_natural_personality))
```

В качестве ответа будет выдано название породы собаки, которую характеризуют представленные в списке свойства, то есть COLLIE (колли).

Литература.

1. Минский М. Фреймы для представления знаний : Пер. с англ. – М.: Энергия, 1979
2. Представление и использование знаний : Пер. с япон. / Под ред. Х. Уэно, М. Исидзука. – М.: Мир, 1989. С. 55-98