

# **Отсечение и его использование.**

*Лекция 4 (Часть 1).*

**Общие случаи использования  
отсечения.**

**Специальности : 230105, 010501**

# Причины введения отсечения.

Отсечение позволяет указать, какие из сделанных ранее выборов не следует пересматривать при возврате по цепочке согласованных целевых утверждений.

## Преимущества введения отсечений.

- Сокращение времени выполнения программы за счет отсутствия новых сопоставлений для целей, не способных более внести ничего нового в решение.
- Уменьшение занимаемого программой объема памяти за счет отсутствия необходимости запоминания точек возврата.

# Синтаксис отсечения.

Использование отсечения в правиле выглядит как вхождение ЦУ с предикатом “!”.

Отсечение :

- не имеет аргументов;
- всегда согласуется с базой данных;
- не допускает повторного согласования;
- имеет побочный эффект, который изменяет процесс последующего возврата.

Изменение процесса последующего возврата заключается в недоступности маркеров некоторых целей.

# Изменение процесса последующего возврата при использовании отсечения.

```
/* Библиотека */
predicates
  reader(symbol).
  book_not_give(symbol,symbol).
  basic_service(symbol).
  addition_service(symbol).
  common_service(symbol).
  service(symbol,symbol).
clauses
/* Данные о читателях */
  reader("Иванов Иван Иванович").
  reader("Петров Петр Петрович").
  reader("Сидоров Сидор Сидорович").
  reader("Смирнов Алексей Анатольевич").
/* Данные о не возвращенных в срок книгах */
  book_not_give("Иванов Иван Иванович",
  "Опыт теории лингвистических моделей").
  book_not_give("Петров Петр Петрович",
  "Лексическая семантика").
  book_not_give(
  "Сидоров Сидор Сидорович",
  "Системы машинного перевода ЭТАП-2").

/* Данные об основных услугах */
  basic_service("Пользование каталогом").
  basic_service("Справочно-
библиографические услуги").
/* Данные о дополнительных услугах */
  addition_service("Абонемент").
  addition_service("Межбиблиотечный
абонемент").
/* Услуги, предоставляемые библиотекой */
  common_service(X):-basic_service(X).
  common_service(X):-addition_service(X).
/* Виды услуг, доступные читателю */
  service(Reader,Service):-
  book_not_give(Reader,Book),
  !,
  basic_service(Service).
  service(Reader,Service):-
  common_service(Service).
goal
  reader("Иванов Иван Иванович"),
  service("Иванов Иван Иванович",Service).
```

# Диаграмма согласования целевого утверждения.



# Эффект отсеечения.

Отсечение в программе “отсекает” путь, представляющий цепочку выполненных доказательств, таким образом, что следующая цель соединяется непосредственно с исходной.

В приведенном примере результат действия отсеечения в правиле для предиката *service* заключается в том, что все цели, выбранные с момента, когда было выбрано это правило, запоминаются в системе как неизменяемые при обратном просмотре. Целевое утверждение *service* называется родительским целевым утверждением для отсеечения, поскольку именно это целевое утверждение привело к использованию правила, содержащего отсеечение.

Если отсеечение встречается в качестве целевого утверждения, то после этого система лишается возможности изменять решения, принятые ею с момента вызова родительского утверждения. Все альтернативы принятым решениям отбрасываются. Попытка вновь доказать согласованность с базой данных любого целевого утверждения между родительским целевым утверждением и отсечением заканчивается неудачей.

# Общие случаи использования отсечения.

- Указание Прологу на то, что найдено нужное правило для заданного целевого утверждения.
- Указание Прологу необходимости немедленного прекращения доказательства согласованности конкретного целевого утверждения без поиска альтернативных решений. В этом случае используется конъюнкция отсечения с предикатом fail, что означает : “если вы дошли до этого места, то вам следует прекратить попытки доказать согласованность данного целевого утверждения.”
- Завершение “порождения и проверки”. Использование механизма возврата для прекращения порождения альтернативных решений. В этом случае отсечение означает : “если вы дошли до этого места, то вы нашли единственное решение задачи и никакой возможности найти другие альтернативные решения нет”

## Подтверждение правильности выбора правила.

Рассмотрим пример - сумма целых чисел от 1 до N.

predicates

sum(integer,integer).

clauses

sum(1,1):-!.

sum(N,S):-N1=N-1,sum(N1,S1),S=S1+N.

В приведенном примере отсечение указывает Прологу на то, что если выбрано первое правило, то больше не следует принимать нового решения относительно выбираемого правила для целевого утверждения *sum*.

Более типичная ситуация - введение дополнительных условий выбора соответствующих правил из БД в процессе согласования. Рассмотрим альтернативный вариант решения последнего примера.

predicates

sum(integer,integer).

clauses

sum(N,1):-N<=1,!.

sum(N,R):-N1=N-1,sum(N1,R1),R=R1+N.



# Предикат not как альтернатива отсечению.

Общий принцип заключается в том, что использование механизма отсечения для указания Прологу на ситуации, когда он выбрал единственно правильное правило, может быть заменено использованием предиката not. not(X) означает, что X недоказуемо как целевое утверждение Пролога.

В качестве примера рассмотрим два приведенных нами варианта определения предиката sum. Вариант 1 :

predicates

sum(integer,integer).

clauses

sum(1,1).

sum(N,R):-not(N=1),N1=N-1,sum(N1,R1),R=N+R1.

Вариант 2 :

predicates

sum(integer,integer).

clauses

sum(N,1):-N<=1.

sum(N,R):-not(N<=1),N1=N-1,sum(N1,R1),R=N+R1.

# Отсечение и fail.

Предикат `fail` является встроенным предикатом Пролога и не имеет аргументов. Выполнение ЦУ `fail` не зависит от значений переменных в правиле, доказательство его согласованности с БД всегда заканчивается неудачей и приводит к включению механизма возврата.

Если `fail` встречается после отсечения, то нормальное выполнение возврата изменится в результате действия механизма отсечения.

Пример использования-в программе вычисления размера налога с использованием “среднего налогоплательщика”.

```
predicates
  wife(symbol,symbol).
  mid_taxp(symbol).
  foreigner(symbol).
  income(symbol,integer).
  salary(symbol,integer).
  cap_income(symbol,integer).
  received_grant(symbol,integer)
```

```
clauses
  wife(mary,ivan).
  foreigner(bill).
```

```
salary(mary,120).
salary(ivan,200).
salary(sergey,150).
salary(oleg,100).
cap_income(bill,150).
cap_income(sergey,200).
received_grant(peter,50).
mid_taxp(X):-foreigner(X),!,fail.
mid_taxp(X):-
  wife(X,Y),income(Y,Income),
  Income>150,!,fail.
```

```
mid_taxp(X):-
  income(X,Income),Income>=50,
  Income<=150.
income(X,Y):-
  received_grant(X,Y),Y<50,!,fail.
income(X,Y):-salary(X,Y).
income(X,Y):-salary(X,Z),
  cap_income(X,W),Y=Z+W.
```

## Замена “отсечения+fail” предикатом not.

Суть использования отсечения в приведенном примере заключается в том, что попытка доказать, что некоторый человек является средним налогоплательщиком, может быть прервана при выполнении некоторого условия (или группы условий). Комбинация “Отсечение+fail” дает возможность остановить поиск альтернатив, прежде чем будет выполнен предикат fail. Мы можем заменить “отсечение+fail” использованием not (на примере Турбо-Пролога) :

```
not_midt(X):-wife(X,Y),income(Y,Income),Income>150.
```

```
mid_taxp(X):-not(foreigner(X)),not(not_midt(X)),  
            income(X,Income),Income>=50,Income<=150.
```

```
income(X,Y):-received_grant(X,Y),not(Y<50).
```

```
income(X,Y):-salary(X,Y).
```

```
income(X,Y):-cap_income(X,Y).
```

```
income(X,Y):-salary(X,Z),cap_income(X,W),  
            Y=Z+W.
```

# Завершение “порождения и проверки”.

ЦУ - генераторы, порождающие все возможные альтернативы и  
ЦУ - контроллеры, проверяющие пригодность решения.

Пример : игра “крестики-нолики”, введение отсечения  
равносильно следующему заявлению : “если ищутся  
вынужденные ходы, то важно найти только первое решение”.

1 2 3	X	0
4 5 6		
7 8 9	X	0 X

# Фрагмент программы игры в “крестики-нолики”

```
field_line([1,2,3]).
field_line([4,5,6]).
field_line([7,8,9]).
field_line([1,4,7]).
field_line([2,5,8]).
field_line([3,6,9]).
field_line([1,5,9]).
field_line([3,5,7]).

arg(_,[],"empty").
arg(1,[H_board|T_board],H_board):-!.
arg(Pos,[_|T_board],Sym):-
    Pos1=Pos-1,arg(Pos1,T_board,Sym).

cross_pos(Pos,Board):-
    arg(Pos,Board,Sym),Sym="X".
zero_pos(Pos,Board):-
    arg(Pos,Board,Sym),Sym="0".
empty_pos(Pos,Board):-
    arg(Pos,Board,Sym),Sym="empty".

thread([Pos,X,Y],Board,Pos):-
    empty_pos(Pos,Board),
    cross_pos(X,Board),
    cross_pos(Y,Board).
thread([X,Pos,Y],Board,Pos):-
    empty_pos(Pos,Board),
    cross_pos(X,Board),
    cross_pos(Y,Board).
```

```
thread([X,Y,Pos],Board,Pos):-
    empty_pos(Pos,Board),
    cross_pos(X,Board),
    cross_pos(Y,Board).
thread([Pos,X,Y],Board,Pos):-
    empty_pos(Pos,Board),
    zero_pos(X,Board),
    zero_pos(Y,Board).
thread([X,Pos,Y],Board,Pos):-
    empty_pos(Pos,Board),
    zero_pos(X,Board),
    zero_pos(Y,Board).
thread([X,Y,Pos],Board,Pos):-
    empty_pos(Pos,Board),
    zero_pos(X,Board),
    zero_pos(Y,Board).

forced_move(Board,Pos):-
    field_line(Field_line),
    thread(Field_line,Board,Pos),!.
```

# Управление “порождением и проверкой” при реализации целочисленного деления.

Другой пример управления “порождением и проверкой” - реализация целочисленного деления :

```
predicates
    int_num(integer).
    div(integer,integer,integer).
clauses
/* Генератор целых чисел */
    int_num(0).
    int_num(X):-
        int_num(Y),X=Y+1.
/* Предикат целочисленного деления */
    div(N1,N2,Result):-
        int_num(Result),
        P1=Result*N2,
        P2=(Result+1)*N2,
        P1<=N1,P2>N1,!.

```

Правило `div` использует предикат `int_num` как генератор всех возможных целых чисел-альтернатив, остальные целевые утверждения выполняют функцию контроллеров. Если убрать отсечение, то любой возврат будет заново инициализировать поиск альтернатив для `int_num`. При этом ни одно значение не было бы правильным результатом деления и генерация целых чисел продолжалась бы до бесконечности.

# Использование отсечений при работе со списками.

Примером использования отсечения для указания Прологу правильности выбора правила для заданного целевого утверждения может служить удаление первого вхождения элемента в список. Ниже представлены для сравнения правила удаления : всех вхождений заданного элемента в список и удаление первого вхождения.

```
/* Удаление всех вхождений  
   некоторого элемента */  
delete(_,[ ],[ ]).  
delete(X,[X|T],M):-delete(X,T,M).  
delete(X,[Y|T],[Y|M]):-  
   X<>Y,delete(X,T,M).
```

```
/* Удаление первого вхождения  
   элемента в список */  
del_first(_,[ ],[ ]).  
del_first(X,[X|T],T):-!.  
del_first(X,[Y|T],[Y|M]):-  
   X<>Y,del_first(X,T,M).
```

Если убрать отсечение из второго правила, то получим правило удаления всех вхождений заданного элемента в список.

## Проблемы, связанные с использованием отсечения.

Если отсечение вводится с целью обеспечения правильности работы программы, то нет гарантии, что появление целевых утверждений иной формы не приведет к непредвиденному результату. В качестве примера рассмотрим измененное определение предиката `append`.

*/\* Исходный вариант \*/*

```
append([ ],L,L).
```

```
append([H1|T1],L2,[H1|T]):-  
    append(T1,L2,T).
```

*/\* Измененный вариант \*/*

```
append1([ ],L,L):-!.
```

```
append1([H1|T1],L2,[H1|T]):-  
    append1(T1,L2,T).
```

Рассмотрим поведение правил `append` и `append1` в случае использования незапланированным способом. Пусть имеются ЦУ `append(X,Y,[1,2,3])` и `append1(X,Y,[1,2,3])`. По аналогии с первым второе ЦУ будет сопоставлено с заголовком первого правила для `append1`, что даст  $X=[ ]$ ,  $Y=[1,2,3]$ , но у `append1` затем встретится отсечение. Это приводит к тому, что альтернативные варианты `append1` будут недоступны, хотя для данного запроса у `append` имеются другие решения :  $X=[1]$ ,  $Y=[2,3]$  ;  $X=[1,2]$ ,  $Y=[3]$  ;  $X=[1,2,3]$ ,  $Y=[ ]$ .



# **Вывод.**

**Надежное использование отсечения возможно лишь в случае наличия полной и достоверной информации о характере использования правил.**

**Если характер использования правил меняется, то необходимо пересмотреть все случаи употребления отсечения.**

# **Литература.**

**Клоксин У., Меллиш К.**

**Программирование на языке**

**Пролог : Пер. с англ. - М.: Мир,**

**1987. С. 84-111**