

프로그래밍 언어론 HW5

B911197 최준영

June 2023

1 Prolog에 대한 설명

Prolog는 논리형 프로그래밍 언어로 3가지의 구성요소로 동작한다. 구성요소에는 사실, 규칙, 질문이 있다.

사실은 객체들 사이의 관계를 정의한다. 규칙을 통해 사실들 사이의 관계를 설정할 수 있다.

마지막으로 프롤로그 프로그램을 시작하면 질문을 할 수 있는데, 사실관계에 대해 묻거나 규칙에 변수를 사용하여 특정 사실에 부합하는 아톰을 알아낼 수 있다.

프롤로그의 데이터 타입은 3가지가 있다. 변수, 아톰, 상수. 프롤로그의 변수는 메모리 공간을 의미하지 않느다. 변수 특정 규칙에 부합하는 사실관계를 전달받기 위해 존재한다.

2 전체적인 코드 작동방식

2.1 hw5a: 삽입정렬

hw5a.pl에 구현한 삽입정렬은 다음과 같이 동작한다.

1. 정렬대상 리스트의 Head를 분리해 가며 가장 마지막 요소부터 빈 리스트에 삽입을 시작한다.

아래사진의 하단을 보면 타깃으로 가장 마지막 요소인 135가 지정됐음을 확인할 수 있다.

```
[trace] ?- sorting([11,33,23,45,13,25,8,135], X).
Call: (10) sorting([11, 33, 23, 45, 13, 25, 8, 135], _2984) ? creep
Call: (11) sorting([33, 23, 45, 13, 25, 8, 135], _3474) ? creep
Call: (12) sorting([23, 45, 13, 25, 8, 135], _3518) ? creep
Call: (13) sorting([45, 13, 25, 8, 135], _3562) ? creep
Call: (14) sorting([13, 25, 8, 135], _3606) ? creep
Call: (15) sorting([25, 8, 135], _3650) ? creep
Call: (16) sorting([8, 135], _3694) ? creep
Call: (17) sorting([], _3738) ? creep
Call: (18) sorting([], _3782) ? creep
Exit: (18) sorting([], []) ? creep
Call: (18) write('target: ') ? creep
target:
Exit: (18) write('target: ') ? creep
Call: (18) write(135) ? creep
```

2. 지정된 타깃을 삽입될 리스트의 요소들과 비교하여 삽입 위치를 결정한다.
아래 사진은 가장 마지막 요소인 135가 비어있는 리스트에 삽입되는 과정이다.
출력 리스트가 [135]임으로 해당과정이 잘 수행되었음을 알 수 있다.

```
Call: (18) insert(135, [], _4136) ? creep
Exit: (18) insert(135, [], [135]) ? creep
Call: (18) write('inserted list: ') ? creep
inserted list:
Exit: (18) write('inserted list: ') ? creep
Call: (18) write([135]) ? creep
[135]
```

배열이 비었지 않을 경우 아래 trace목록처럼 [135]리스트의 135와 값을 비교후 삽입된다.

```
Exit: (17) nl ? creep
Call: (17) insert(8, [135], _4802) ? creep
Call: (18) 8=<135 ? creep
Exit: (18) 8=<135 ? creep
Exit: (17) insert(8, [135], [8, 135]) ? creep
Call: (17) write('inserted list: ') ? creep
inserted list:
Exit: (17) write('inserted list: ') ? creep
Call: (17) write([8, 135]) ? creep
[8, 135]
```

[8, 135]가 잘 출력되는 것을 확인할 수 있다.
삽입하려는 요소가 중간에 삽입되는 경우는 아래 사진과 같은 과정을 거친다.

```
25
Exit: (16) write(25) ? creep
Call: (16) nl ? creep

Exit: (16) nl ? creep
Call: (16) insert(25, [8, 135], _5562) ? creep
Call: (17) 25=<8 ? creep
Fail: (17) 25=<8 ? creep
Redo: (16) insert(25, [8, 135], _5706) ? creep
Call: (17) 25>8 ? creep
Exit: (17) 25>8 ? creep
Call: (17) insert(25, [135], _5696) ? creep
Call: (18) 25=<135 ? creep
Exit: (18) 25=<135 ? creep
Exit: (17) insert(25, [135], [25, 135]) ? creep
Exit: (16) insert(25, [8, 135], [8, 25, 135]) ? creep
Call: (16) write('inserted list: ') ? creep
inserted list:
Exit: (16) write('inserted list: ') ? creep
Call: (16) write([8, 25, 135]) ? creep
[8, 25, 135]
```

먼저 25는 8보다 큼으로 [8,135]리스트를 [135]리스트로 분해한 후 삽입과정을 분해한 리스트로 다시 수행한다.

25는 135보다 작음으로 [25,135] 리스트가 만들어진다. 그 후 제귀적 호출종료 시점으로 돌아와 [8,25,135] 리스트로 재구성된다.

3. 모든 요소가 해당과정을 거치게 되면 빈리스트가 다 채워지게되 정렬된 리스트가 완성되고 출력된다.

2.2 hw5b: N-Queen problem

구현한 코드는 다음과 같은 순서로 작동한다.

1. 한행에는 하나의 퀸밖에 존재할 수 없음을 전제한다.
2. N*N체스판에 대해 1부터 N까지의 요소를 포함하는 리스트를 만든다.
3. 해당리스트를 permutation규칙을 사용하여 임의의로 정렬된 리스트를 획득한다.
4. 해당리스트는 퀸들의 배치를 의미함으로 모든 퀸을 순회하며 다른 퀸들과 배치 유효성을 확인한다.
5. 유효성 테스트를 통과한 리스트는 배치가 가능한 리스트임으로 출력된다.

```
[trace] ?- n_queen(4,X).
Call: (10) n_queen(4, _2930) ? creep
Call: (11) lists:numlist(1, 4, _4382) ? creep
Exit: (11) lists:numlist(1, 4, [1, 2, 3, 4]) ? creep
Call: (11) lists:permutation([1, 2, 3, 4], _2930) ? creep
Exit: (11) lists:permutation([1, 2, 3, 4], [1, 2, 3, 4]) ? creep
Call: (11) checkcombination([1, 2, 3, 4]) ? creep
```

N=4에 대해 1,2,3,4 리스트를 만든다. 그 후 첫번째 조합인 [1,2,3,4]의 유효성 검사를 실시한다.

```
Call: (12) checkconflict(1, [2, 3, 4]) ? creep
Call: (13) 1=\=2 ? creep
Exit: (13) 1=\=2 ? creep
Call: (13) _7598 is 1 ? creep
Exit: (13) 1 is 1 ? creep
Call: (13) 1+1=\=2 ? creep
Fail: (13) 1+1=\=2 ? creep
Fail: (12) checkconflict(1, [2, 3, 4]) ? creep
Fail: (11) checkcombination([1, 2, 3, 4]) ? creep
```

첫번째퀸의 배치 1에대해 확인하는 과정에서 $1+1=\bar{2}$ 부분이 실패한 것을 확인할 수 있다.

1+1은 1행의 대각선행을 의미하며 해당 행에 현재 확인하는 퀸(2)이 위치하고 있음으로 실패한다.

현재 조합은 실패했음으로 다음조합의 유효성을 검사한다.

아래사진은 [1,2,4,3]조합의 유효성을 검사하려고 하고있다.

```
Redo: (11) lists:permutation([1, 2, 3, 4], [1, 2, 3, 4]) ? creep
Exit: (11) lists:permutation([1, 2, 3, 4], [1, 2, 4, 3]) ? creep
Call: (11) checkcombination([1, 2, 4, 3]) ? creep
Call: (12) checkconflict(1, [2, 4, 3]) ? creep
```

모든 유효성 검사를 통과한 조합은 X변수로 전달되게 된다.

```
Call: (15) checkconflict(1, [], 2) ? creep
Exit: (15) checkconflict(1, [], 2) ? creep
Exit: (14) checkconflict(1, [3]) ? creep
Call: (14) checkcombination([3]) ? creep
Call: (15) checkconflict(3, []) ? creep
Exit: (15) checkconflict(3, []) ? creep
Call: (15) checkcombination([]) ? creep
Exit: (15) checkcombination([]) ? creep
Exit: (14) checkcombination([3]) ? creep
Exit: (13) checkcombination([1, 3]) ? creep
Exit: (12) checkcombination([4, 1, 3]) ? creep
Exit: (11) checkcombination([2, 4, 1, 3]) ? creep
Exit: (10) n_queen(4, [2, 4, 1, 3]) ? creep
X = [2, 4, 1, 3] .
```

3 작성한 코드 설명

3.1 hw5a.pl

% 삽입하려는 리스트가 빈 경우 요소를 빈 리스트에 삽입
insert(Element, [], [Element]).

% 삽입하려는 리스트의 헤드보다 Element값이 작다면 리스트의 헤드에 Element를 삽입
insert(Element, [Head|Tail], [Element, Head|Tail]) :-
 Element < Head.
insert(Element, [Head|Tail], [Head|Result]) :-
 % Element가 헤드보다 값이 크다면 insert를 Head를 제외한 리스트를 사용하여 호출
 Element > Head,
 insert(Element, Tail, Result). % 헤드를 제외한 리스트에서 삽입 위치를 탐색

% 리스트의 끝 요소부터 빈 리스트에 삽입되는 과정으로 정렬이 진행된다.
sorting([], []).

```
sorting([Head|Tail], Result) :-  
    sorting(Tail, Rest),  
    write('target: '),  
    write(Head), nl,    % 삽입의 대상이 되는 요소를 출력  
    insert(Head, Rest, Result),  
    write('inserted list: '),  
    write(Result), nl. % 삽입과정이후 변경된 리스트 출력
```

3.2 hwb.pl

```
% 특정 퀸에 대해 리스트의 다른 퀸들과 비교하여 해당 위치에 배치가 가능한지 확인하는 규칙  
checkconflict(_, []). % 비교대상이 없을 경우, 해당 조합이 성공적이라는 의미이다.  
checkconflict(Queen, [Row|OtherQueens]) :- % 최초로 유효성 검사를 시작한다.  
    Queen =\= Row,  
    DiagonalDistance is 1, % 다음퀸과의 대각선 거리를 의미한다.  
    Queen + DiagonalDistance =\= Row, % 대각선거리에 비교하는 퀸의 위치할 경우 두 퀸이 대각선상  
에 위치함으로 false  
    Queen - DiagonalDistance =\= Row,  
    NewDistance is DiagonalDistance + 1, % 다음퀸은 비교대상 퀸과의 거리가 1만큼 떨어짐으로 대각거  
리를 1만큼 증가시킨다.  
    checkconflict(Queen, OtherQueens, NewDistance). % 유효성 검증 성공후 다음 퀸으로 이동  
  
checkconflict(_, [], _). % 더이상 비교할 퀸이 없는 경우, 즉 모든 퀸에 대해 해당 위치에 배치가 가능  
checkconflict(Queen, [Row|OtherQueens], DiagonalDistance) :-  
    Queen =\= Row,  
    Queen + DiagonalDistance =\= Row,  
    Queen - DiagonalDistance =\= Row,  
    NewDistance is DiagonalDistance + 1,  
    checkconflict(Queen, OtherQueens, NewDistance). % 다음퀸으로 이동  
  
checkcombination([]).  
checkcombination([Queen|OtherQueens]) :-  
    checkconflict(Queen, OtherQueens), % 전달받은 리스트가 가장 앞쪽 퀸을 다른 퀸들과 비교하  
며 유효성 확인  
    checkcombination(OtherQueens). % 제귀적 호출을 통해 다른 퀸들도 유효성을 검사  
  
n_queen(N, Queens) :-  
    numlist(1, N, Rows), % 1...N범위의 요소를 가진 리스트를 생성합니다.  
    permutation(Rows, Queens), % 전달한 리스트를 가능한 경우의 수만큼 랜덤하게 정렬합니다.  
    checkcombination(Queens). % 랜덤하게 정렬된 리스트가 유효한지 확인합니다.
```

4 코드실행결과

4.1 hw5a.pl

```
[B911197@linux2 hw5c]$ swipl hw5a.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- sorting([11,33,23,45,13,25,8,135], X).
target: 135
inserted list: [135]
target: 8
inserted list: [8,135]
target: 25
inserted list: [8,25,135]
target: 13
inserted list: [8,13,25,135]
target: 45
inserted list: [8,13,25,45,135]
target: 23
inserted list: [8,13,23,25,45,135]
target: 33
inserted list: [8,13,23,25,33,45,135]
target: 11
inserted list: [8,11,13,23,25,33,45,135]
X = [8, 11, 13, 23, 25, 33, 45, 135]
```

```
[B911197@linux2 hw5c]$ swipl hw5a.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- sorting([83,72,65,54,47,33,29,11], X).
target: 11
inserted list: [11]
target: 29
inserted list: [11,29]
target: 33
inserted list: [11,29,33]
target: 47
inserted list: [11,29,33,47]
target: 54
inserted list: [11,29,33,47,54]
target: 65
inserted list: [11,29,33,47,54,65]
target: 72
inserted list: [11,29,33,47,54,65,72]
target: 83
inserted list: [11,29,33,47,54,65,72,83]
X = [11, 29, 33, 47, 54, 65, 72, 83]
```

4.2 hw5b.pl

```
[B911197@linux2 hw5c]$ swipl hw5b.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- n_queen(10,X).
X = [1, 3, 6, 8, 10, 5, 9, 2, 4|...]. 

?- n_queen(8,X).
X = [1, 5, 8, 6, 3, 7, 2, 4] . 

?- n_queen(4,X).
X = [2, 4, 1, 3] ;
X = [3, 1, 4, 2] . 

?- halt.
[B911197@linux2 hw5c]$
```

5 어려웠던 점들

처음접해보는 프로그래밍 방식이라 처음에 어떻게 구현을 해야할지 막막했다.

변수도 기존의 프로그래밍 언어와는 달리 특정 사실에 결과값을 전달받는 구조는 직관적으로 이해하기 힘들었다. 과제에 제시된 문제들은 문제들은 재귀적 호출을 사용해야하는데 return이라는 개념이 없는 언어여서 코드구현이 어려웠다.

기존의 언어들은 인덱싱을 통해 콜렉션 요소에 간편하게 접근이 가능했지만 프롤로그는 그렇지 않아 힘들었다. 플로로그는 반복문을 모두 재귀적으로 구현해야해서 직관성이 많이 떨어진다. 그래서 힘들었다.