1. Prolog 입문

Prolog(Programming in Logic)는 논리 프로그래밍 언어를 대표하는 언어이다. Prolog는 술어계산법(Predicate Calculus)이라고 하는 논리학에 기반하여 컴퓨터 프로그램 작성이 가능하게 위하여 개발된 것이다. 1970년대에 Aix-Marseille 대학의 A. Colmerauer과 P. Roussel 그리고 Edinburgh 대학의 R. Kowalski 등이 공동으로 Prolog를 설계하였다.

Prolog 프로그램은 사실과 규칙으로 이루어지고, 실행순서를 기술하지 않는다. 따라서 Prolog는 비절차적 프로그래밍 언어의 일종이며 초고급언어(혹은 5세대 언어)라고 일컬어진다. 프로그램 실행의 결과는 풀려는 문제를 질의하여 얻은 답으로 나타난다. Prolog는 함수 프로그래밍 언어인 Lisp(List Processing)와 함께 인공지능 분야에서 효과적으로 사용하고 있다.

1.1 Prolog 설치

프로그래밍 언어를 사용하여 프로그램을 작성하기 위해서는 언어에 대한 번역기를 포함한 프로그래밍 시스템이 필요하다. 여기서는 UNIX는 물론 MS-Windows와 MacOS 환경에서도 사용할 수 있는 SWI-Prolog를 소개한다. SWI-Prolog는 무료로 배포되므로 자유롭게 이용할 수 있다. (실험실습에 사용할 설치 파일은 UClass에서 내려받아야 한다.)

(1) SWI-Prolog Home
http://www.swi-prolog.org/

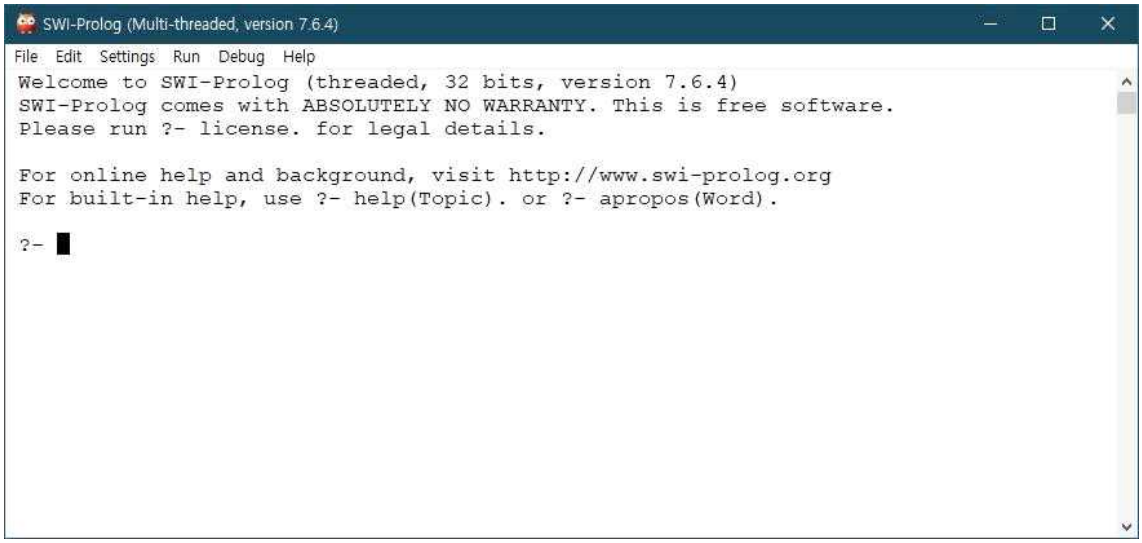
(2) SWI-Prolog 설치

① 안정된 배포판(Stable release)
② SWI-Prolog/XPCE for MS-Windows
③ swipl-w32-764.exe (실험실습에 사용할 설치 파일)

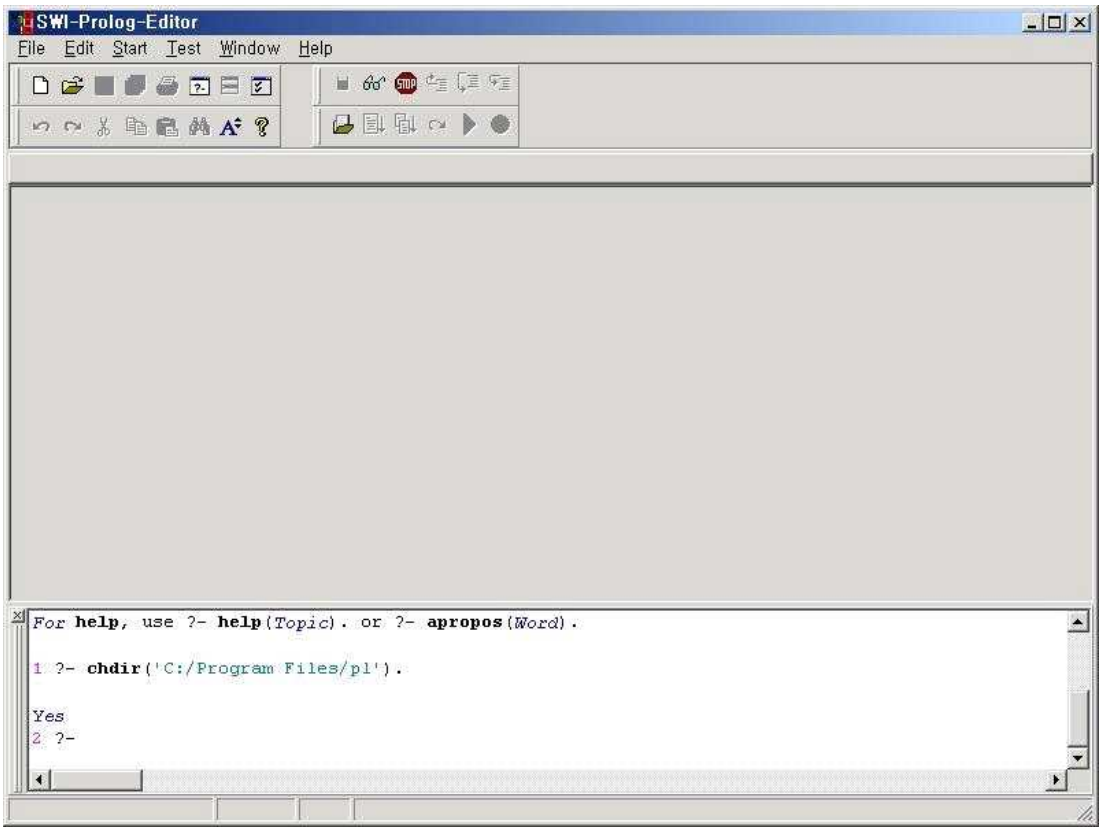
(3) SWI-Prolog 프로그램 실행

① Prolog 프로그램을 작성
가. 메모장을 이용한다.
나. 확장자는 pl이다.
② Prolog 프로그램을 두 번 클릭한다.

SWI-Prolog 설치는 매우 간단하다. swipl-w32-764.exe 파일을 실행하여 기본 옵션으로 설치한다. 시작 메뉴에 생긴 SWI-Prolog를 실행하면 다음과 같은 화면을 볼 수 있다.



현재 환경에서 프로그램을 작성하여 실행하기에는 불편하다. 따라서 SWI-Prolog 프로그래머에게 IDE(통합개발환경)을 제공하는 SWI-Prolog-Editor를 설치하는 것이 바람직하다. SWI-Prolog-Editor는 SWI-Prolog가 먼저 설치되어 있어야 동작한다. (실험 실습에 사용할 설치 파일: “SWI-Prolog-Editor Setup.exe”)



1.2 Prolog 프로그래밍 시작

SWI-Prolog-Editor를 실행했을 때 앞의 화면을 볼 수 있다면 Prolog 프로그래밍을 하기 위한 모든 준비가 끝난 것이다. File 메뉴에서 New를 선택하면 소스 코드를 작성할 수 있는 창이 뜬다. 다음과 같은 내용을 입력하고 ex1.pl 이라는 이름으로 저장한다.

```
man(socrates).
mortal(X) :- man(X).
```

Prolog 프로그램은 사실(Fact)과 규칙(Rule)으로 이루어져 있다. 위의 프로그램은 man(socrates)라는 사실 한 개와 mortal(X) :- man(X)라는 규칙 하나로 이루어져 있다. 각 문장은 마침표로 끝난다.

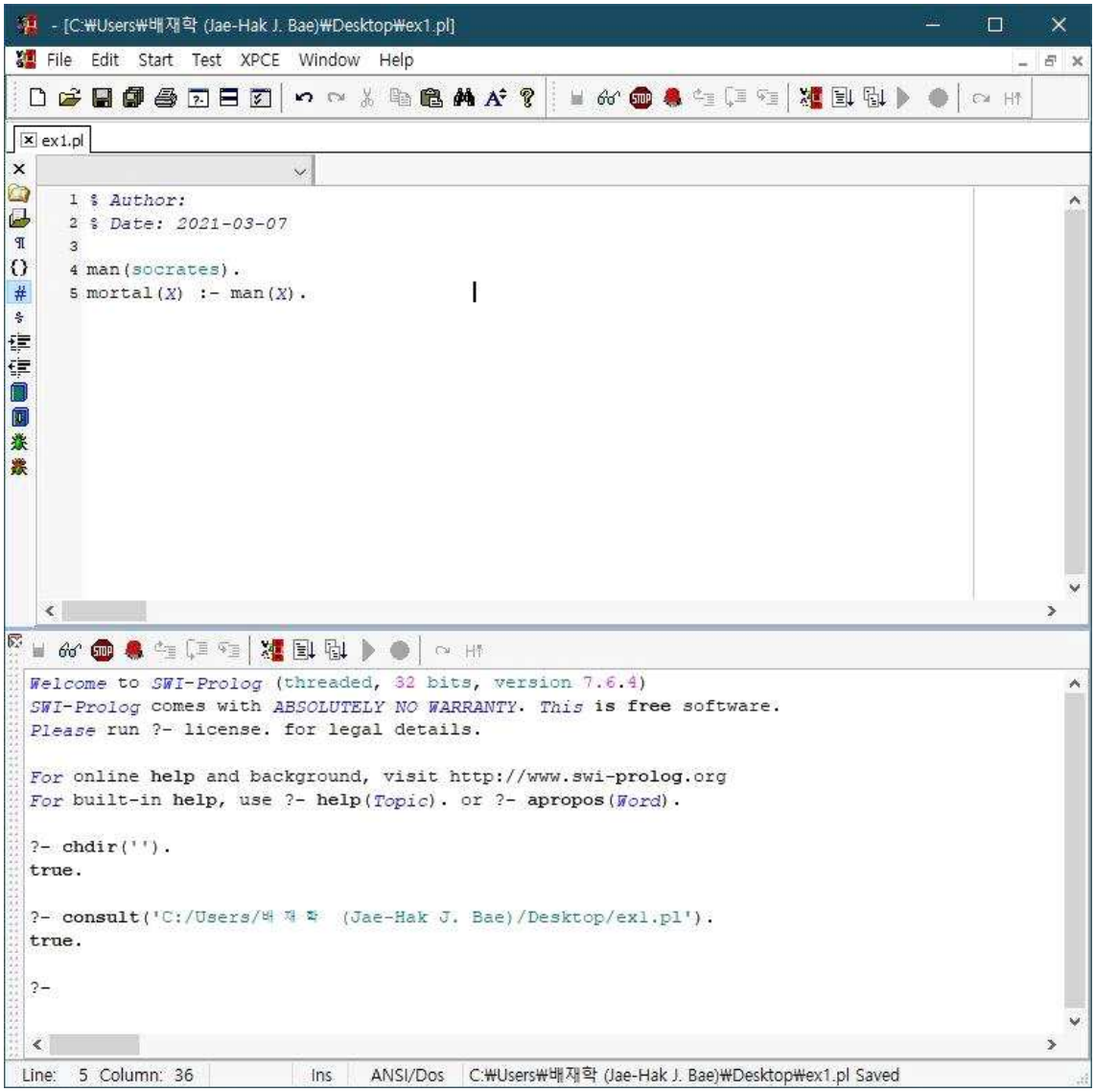
```
man(socrates).
```

이 문장은 소크라테스가 사람이라는 사실을 표현한다. socrates를 Socrates 라고 표기하면 안 된다. Prolog는 대문자로 시작하는 이름은 변수로 인식하기 때문에, 일반적인 명사로 사용하고 싶을 때에는 소문자로 시작하는 이름을 써야한다.

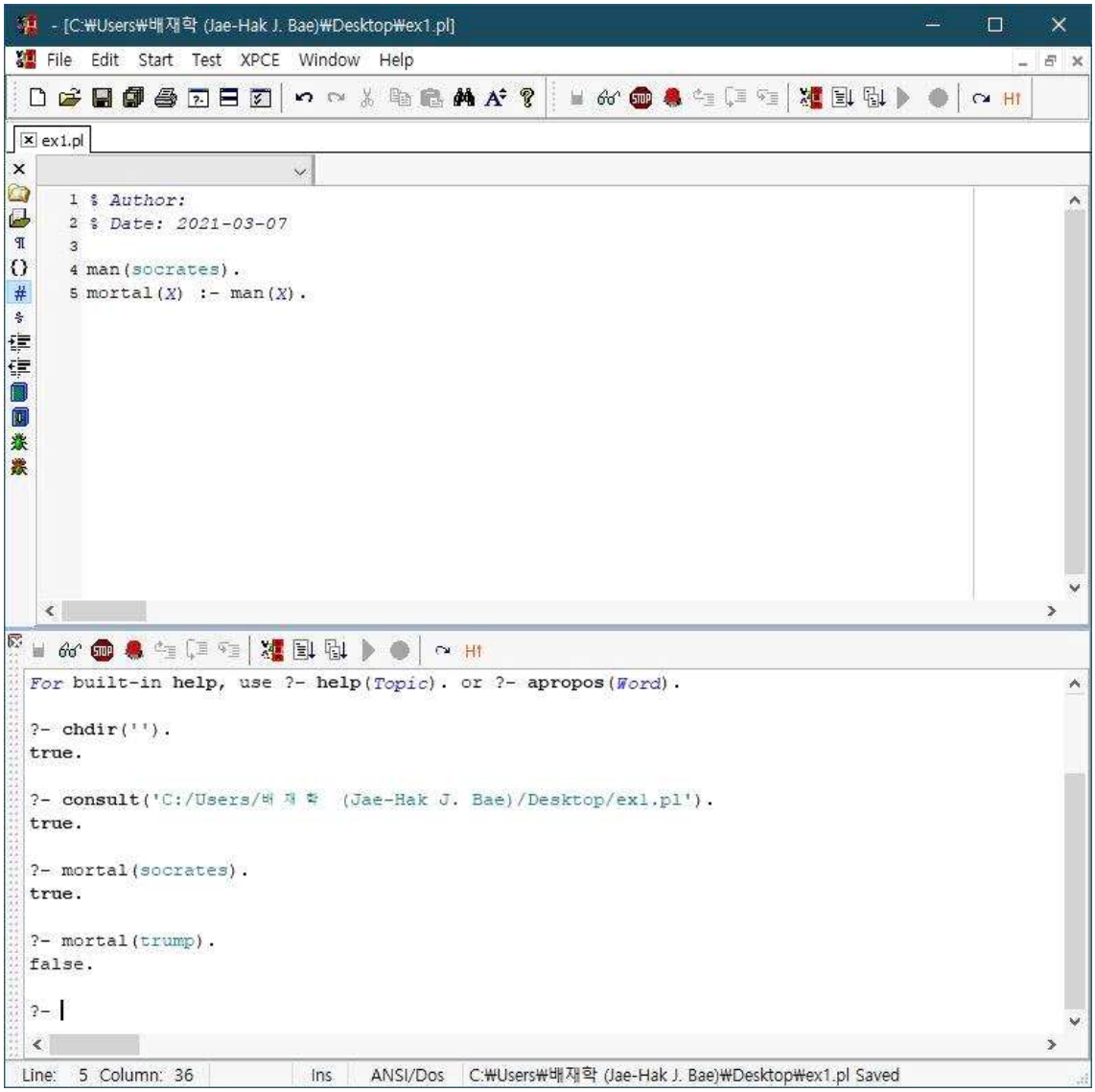
```
mortal(X) :- man(X).
```

이 문장은 규칙의 예이다. 변수 X의 값이 사람이라면, 그 X는 언젠가는 죽는다는 것을 표현한다. 규칙은 "결론 :- 조건" 의 구조를 갖는데, ':-' 는 논리기호 '←'를 대신하고 있다.

이제 F9 키를 누르면 다음과 같이 두 문장으로 구성된 앞의 프로그램이 실행된다.



아래 창에 나타난 ?- 기호는 Prolog 가 질문을 받을 준비가 되었다는 의미의 프롬프트이다. 우리의 관심사는 '모든 사람은 죽는다', '소크라테스는 사람이다'라는 두 명제로부터 '소크라테스는 죽는다'라는 결론이 도출되는가 하는 것이다. 아래 창에 'mortal(socrates).'라고 입력한다.



Prolog는 true라는 대답을 내놓았다. 즉 '소크라테스는 죽는다'라는 명제가 참이라는 것이다. '?- mortal(trump).'로 표현한 '트럼프가 죽는가'라는 질문에 Prolog는 false라는 대답을 하였다. 트럼프가 사람이라는 사실이 없기 때문에, 질문에 대한 답이 이렇게 나왔다.

1) 사실

사실(Fact)은 객체 사이에 성립하는 관계를 표현한다. 'John이 Mary를 좋아한다.'는 사실을 Prolog에서는 다음과 같이 표현한다.

```
likes(john, mary).
```

이 때 다음과 같은 사항들이 중요하다:

① 모든 관계와 객체들의 이름은 반드시 소문자로 써야 한다.
② 관계를 먼저 쓰고, 객체들은 쉼표로 구분하며, 괄호로 묶어야 한다.
③ 끝은 나타낸 점 '.'이 사실의 끝에 나타나야 한다.

다음에 사실에 대한 여러 예를 그 해석과 함께 보였다:

valuable(gold).	금은 귀중하다.
female(jane).	Jane은 여성이다.
owns(john, gold).	John이 금을 소유하고 있다.
father(john, mary).	John은 Mary의 아버지이다.
gives(john, book, mary).	John이 Mary에게 책을 주었다.

2) 질문

알려진 사실이 있으면, 그에 대한 여러 질문(Question)을 할 수 있다. Prolog 질문은 그 앞에 특별한 기호가 있다는 것을 제외하고는, 사실과 그 형태가 같다.

```
?- owns(mary, book).
```

Prolog는 질문을 받으면 주어진 사실로 이루어진 데이터베이스를 탐색하여 사실과 부합하면 Yes라고 반응하고 그렇지 않으면 No라고 반응한다.

<예제 1> 다음과 같이 프로그램을 입력하여 실행을 확인하라.

```
likes(joe, fish).
likes(joe, mary).
likes(mary, book).
likes(john, book).
```

?- likes(joe, money).
No
?- likes(mary, joe).
No
?- likes(mary, book).
Yes

3) 변수

사실을 구성하는 객체가 무엇인지를 알기 원할 때는 변수(Variable)를 사용한다.

```
?- likes(john, X).
```

변수 X에 해당하는 여러 개의 객체가 있을 경우에는 ";"를 입력하여 조건을 만족하는 다른 객체를 계속 찾게 할 수 있다.

<예제 2> 다음과 같이 프로그램을 입력하여 실행을 확인하라.

```
likes(joe, fish).
likes(mary, book).
likes(mary, flower).

?- likes(joe, X).
X = fish
Yes
?- likes(mary, X).
X = book;
X = flower;
No
```

4) 논리곱

단순명제를 논리곱(And)을 나타내는 ','로 연결하여 합성명제로 질문을 할 수 있다.

```
?- likes(john, mary), likes(mary, john).
```

논리곱을 이용한 질문에 변수를 사용할 수도 있다.

```
?- likes(john, X), likes(mary, X).
```

<예제 3> 다음과 같이 프로그램을 입력하여 실행을 확인하라.

```
likes(mary, food).
likes(mary, wine).
likes(john, mary).
likes(john, wine).

?- likes(mary, john), likes(john, mary).
No
?- likes(mary, X), likes(john, X).
X = wine
Yes
```

5) 규칙

규칙(Rule)은 어떤 사실이 다른 사실들의 모임에 의존한다는 것을 나타낸다. 예를 들어 'John은 포도주를 좋아하는 사람은 누구나 좋아한다.'라는 것을 표현하고 싶을 때 다음과 같이 Prolog 규칙을 쓸 수 있다:

```
likes(john, X) :- likes(X, wine).
```

규칙을 나타내는 기호인 ':-'는 if로 발음하고 논리기호 '←'로 해석하면 된다. 또 다른 예로서 'John은 포도주를 좋아하는 여자를 좋아한다.'는 다음 규칙으로 표현할 수 있다.

```
likes(john, X) :- female(X), likes(X, wine).
```

<예제 4> 다음과 같이 프로그램을 입력하여 실행을 확인하라.

```
male(albert).
male(edward).
female(alice).
female(victoria).
```

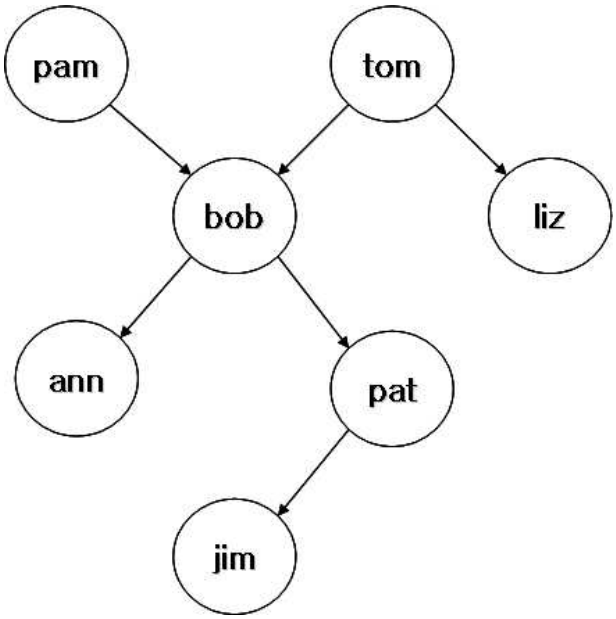
```
parents(edward, victoria, albert).
parents(alice, victoria, albert).
sister_of(X, Y) :- female(X), parents(X, M, F), parents(Y, M, F).

?- sister_of(alice, edward).
Yes
?- sister_of(alice, X).
X = edward
Yes
```

1.3 Prolog 실험 시작

1) 가족관계 계산

가족관계를 Prolog로 표현하고 관계연산의 실험을 생각한다[Prolog for AI]. 다음 그림에서 화살표는 부모와 자녀 관계를 나타낸다.



각 화살표에 대응하는 Prolog 사실은 다음과 같다.

```
parent(pam, bob).
parent(tom, bob).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).
```

여기에 자녀 개념 정의에 대한 규칙을 추가한다.

```
child(X, Y) :- parent(Y, X).
```

parent(A, B)라는 술어(Predicate)는 'A가 B의 부모이다.'를 나타내는 것으로 사용하

였다. ‘child(X, Y) :- parent(Y, X).’는 ‘Y가 X의 부모이면 X는 Y의 자식이다.’라는 뜻을 표현하는 규칙이다. 이제 다음과 같은 질문을 던지고 그 답을 확인하자.

```
?- parent(tom, liz).
Yes
?- child(pat, bob).
Yes
?- child(bob, pat).
No
```

한 걸음 더 나아가 질문에 변수를 포함시킨 경우를 생각한다. 다음의 예를 보자.

```
?- parent(pam, X).
X=bob ;
No
?- child(bob, X).
X=pam ;
X=tom ;
No
```

’pam은 누구의 부모인가?’ 라는 질문에 ’누구란 bob이다’라는 답이 나왔다. 여기서 ‘;’를 입력하면 Prolog는 pam을 부모로 하는 또 다른 답을 찾는다. bob 외에는 pam을 부모로 하는 사람이 없기에 No가 출력된다. child(bob, X)라는 질문에는 Prolog가 bob 의 부모가 누구인지를 찾는다. X=pam 이라는 답 외에도 X=tom 이라는 또 다른 대안이 있으므로 두 가지 답을 출력한다.

다음은 자녀 관계의 모든 경우를 찾는 예이다.

```
?- child(X, Y).
X = bob
Y = pam ;
X = bob
Y = tom ;
X = liz
Y = tom ;
X = ann
Y = bob ;
```

X = pat
Y = bob ;
X = jim
Y = pat ;
No

이제 조부모(할아버지와 할머니) 관계를 나타내는 술어인 grandparent를 정의한다. 조부모는 부모의 부모이다. 이를 Prolog로 표현하기 위해서는 AND 를 나타내는 ‘,’(쉼표)가 필요하다.

grandparent(X, Y) :- parent(X, Z), parent(Z, Y).

앞의 프로그램에 grandparent를 추가하고 tom의 손자를 찾는다.

?- grandparent(tom, X).
X=ann ;
X=pat ;
No

ann과 pat은 bob의 자식이고, bob은 tom의 자식이므로 위와 같은 결과가 나왔다. grandchild를 정의하고 싶다면 앞서 본 child와 비슷한 방식으로 선언할 수 있다. 한편 형제 관계를 나타내는 sibling도 정의한다.

sibling(X, Y) :- parent(Z, X), parent(Z, Y).
?- sibling(ann, pat).
Yes

다음과 같은 질문을 생각하자.

?- sibling(ann, ann).
Yes

ann과 ann, 즉 동일 인물에 대해서 형제인가를 물었는데 Yes 라는 답이 나왔다. sibling(X, Y)에서 변수 이름이 다르다고 해서 변수 X와 Y가 다른 값이어야 한다는 제약은 없다. 상식에 맞게 sibling 정의를 다음과 같이 수정한다.

sibling(X, Y) :- parent(Z, X), parent(Z, Y), X \= Y.

?- sibling(ann, pat).
Yes
?- sibling(ann, ann).
No

'\='는 두 값이 다르면 참값을 가지는 연산자이다. 반대로 같은가를 검사하려면 '=' 를 사용한다. AND 조건은 ',' 를 사용하면 나타낼 수 있듯이, OR 조건은 ';'을 사용하면 나타낼 수 있다. 연산의 우선순위는 AND가 높다.

2) 계승 계산

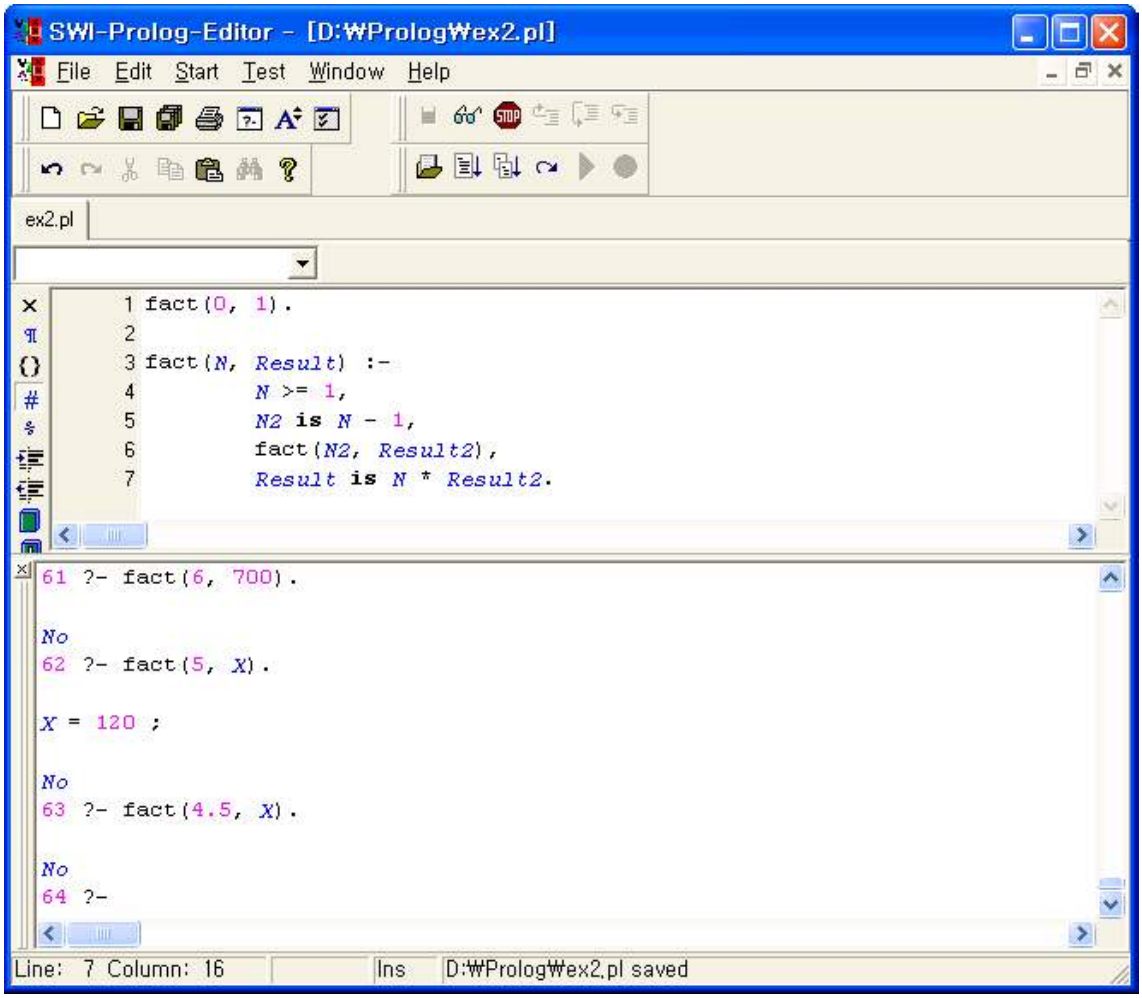
N!(N 계승)은 다음과 같이 재귀적으로 정의할 수 있다.

- 0! = 1
- N! = N * (N - 1)! (N >= 1 일 때)

예를 들어 5! = 5 * 4!이고, 4! = 4 * 3!이고, ..., 1! = 1 * 0!, 0! = 1이므로 결국 5! = 5 * 4 * 3 * 2 * 1이 되어 우리가 알고 있는 계승의 정의와 같다는 것을 알 수 있다. 앞의 정의에 대한 Prolog 프로그램을 본다.

```
fact(0, 1).
fact(N, Result) :-
N >= 1,
N1 is N - 1,
fact(N1, Result1),
Result is N * Result1.
```

수치적인 계산 결과를 변수에 저장할 때 Prolog에서는 '=' 대신 'is'를 쓴다. 만약 '='를 사용하게 되면 N2에 'N - 1'의 계산 결과가 할당되는 것이 아니라 N2와 "N - 1" 두 항을 일치시키는 연산이 일어난다. 계승관계 계산결과가 다음에 있다.



시스템 실험

1. Getting Started

Once Prolog has started up it displays the prompt

?-

which indicates that the interpreter is waiting for a command from the user. All commands must end with a period. For example, the command

?- integer(3.4).

returns the answer no because 3.4 is not an integer. A command is usually called a *goal* or a *query*. To exit the interpreter type *control-D*—press the control key and the D key at the same time.

2. An Introductory Example

A Prolog program is a set of facts or rules called definite clauses. We'll usually refer to them as clauses. The example program that follows describes some family relationships. We'll use the predicates "par" and "grand" with the following meanings.

par(X, Y) means that X is a parent of Y.
grand(X, Y) means that X is a grandparent of Y.

Now we'll list the program, which consists of some parent facts together with a rule that defines the grandparent relationship in terms of parents. Note that a comment is signified by the character % followed by any sequence of characters up to the end of the line. Another way to comment is to place any sequence of characters, including new lines, between the symbols /* and */.

```
% Here is a set of facts describing parental relationships.
par(lloyd, james).
par(lloyd, janet).
par(ruth, james).
par(ruth, janet).
par(emma, lloyd).
par(katherine, ruth).
par(adolph, lloyd).
par(edgar, ruth).

% The grandparent relationship. Any rule of the form
% A :- B, C is read, "A is true if B is true and C is true."
grand(X, Z) :- par(Y, Z), par(X, Y).
```

Now, suppose that you have entered this program into a file named *familyTree*. To read in the program type the following command.

?- [familyTree].

Once the program has been read in it won't do anything until it is presented with a goal.

3. Loading Information

To read in the contents of a file named filename type

?- [filename].

and hit return. If the file name contains characters other than letters or digits, then put single quotes around the filename. For example, if the name of the file is file.p, then type

?- ['file.p'].

will load the file named *file.p*.
You can read in several files at once. For example, to read in files named *foo*, *goo*, and *moo* type

?- [foo, goo, moo].

Sometimes it may be useful to enter a few clauses directly from the a terminal to test something or other. In this case you must type the command

?- [user].

and hit return. The prompt

|:

will appear to indicate that the interpreter is waiting for data. To exit the entry mode type *Control-D*, which we''ll signify by writing *^D*. For example, to enter the two statements p(a, b) and q(X, Y) :- p(X, Y) type the following statements.

?- [user].

|: p(a, b).
|: q(X, Y) :- p(X, Y).
|: end_of_file.

4. Listing Clauses

To list the clauses of a Prolog program type the command

?- listing.

To list only clauses beginning with predicate p type the command

?- listing(p).

To list clauses beginning with predicates p, q, and r type the command

?- listing([p, q, r]).

5. Using Unix Commands

UNIX commands can be executed using the system predicate. For example, to edit the file named filename with the vi editor, type

?- system(''vi filename'').

6. Tracing Note

To interactively trace each step of a computation type the trace command.

?- trace.

Now the execution of any goal will stop after each step. The names of the computation steps are from the set {call, exit, redo, fail}. To continue the computation you must react in one of several ways. For example, here are some of the options that are available.

To “creep” to the next step hit return.

To “leap” to the end of the computation, type *l* and hit return.
To list the menu of available options type *h* and hit return.

You can switch off tracing by typing the notrace command.

?- notrace.

7. Spying Note

It is usually not necessary to creep through every execution step of a program. Spy-points make it possible to stop the execution at predicates of your choice. For example, to stop the execution at each use of the predicate *p*, type the goal

?- spy p.

You can set more than one spy-point. For example, if you want to set spy-points for predicates *p*, *q*, and *r*, type the goal

?- spy [p, q, r].

Now you can “leap” between uses of spy-points or you can still creep from step to step. To “leap” to the next use of a spy-point, type *l* and hit return.

You can remove spy-points too. For example, to remove p as a spy-point, type the nospy goal

?- nospy p.

To remove *p*, *q*, and *r* as spy-points type the goal

?- nospy [p, q, r].

To remove all spy-points type the goal

?- nospyall.