

CONCEPTS OF
PROGRAMMING LANGUAGES

Chapter 16

Logic Programming Languages



ROBERT W. SEBESTA

12/E

ISBN 0-321-49362-1

Chapter 16 Topics

- 서론
- 술어 해석학의 간단한 소개
- 술어 해석학과 정리 증명
- 논리형 프로그래밍의 개관
- **Prolog의 기원**
- **Prolog의 기본**
- **Prolog의 결점**
- 논리형 프로그래밍의 응용

Introduction

- 논리언어에서는 프로그램을 기호 논리의 형식으로 표현
- 논리적 추론과정 사용하여 결과를 도출
- 논리형 프로그램은 절차적이라기보다 선언적임:
 - Only specification of *results* are stated
(not detailed *procedures* for producing them)

명제(Proposition)

- 참이거나 거짓인 논리적 문장
 - 객체와 객체 상호간의 관계로 구성

기호 논리(Symbolic Logic)

- 형식 논리의 다음 세가지 기본 요구사항에 사용될 수 있음 :
 - 명제 표현
 - 명제 사이의 관계 표현
 - 참이라고 추정되는 다른 명제로부터 새로운 명제가 추론되는 방법의 기술
- 논리형 프로그래밍을 위해 사용된 기호 논리의 특별한 형식을 술어 해석학(*predicate calculus*)이라 함

객체 표현

- 명제에서의 객체는 상수나 변수인 간단한 항으로 표현됨
- 상수: 객체를 표현하는 기호
- 변수: 각기 다른 시간에 다른 객체를 표현할 수 있는 기호
 - 명령형 언어의 변수와 다름
- 기본 명제(*Atomic proposition*)는 복합 항으로 구성됨

복합 항의 구성요소

- **복합 항**: 수학적 관계의 한 원소이며, 수학 함수와 같은 형식으로 작성
- 복합 항은 두 부분으로 구성
 - 작용자(Functor): 관계를 명명하는 함수 기호
 - Ordered list of parameters (tuple)
- **Examples:**

```
student(john)
```

```
like(seth, OSX)
```

```
like(nick, windows)
```

```
like(jim, linux)
```

명제의 형식

- 명제는 두 가지 형식으로 나타낼 수 있다:
 - *Fact*: 명제가 참으로 정의되는 모드
 - *Query*: 명제의 진리 값이 결정되어야 하는 모드
- 복합 명제(Compound proposition):
 - Have two or more atomic propositions
 - Propositions are connected by operators

Logical Operators

Name	Symbol	Example	Meaning
부정(negation)	\neg	$\neg a$	not a
논리곱(conjunction)	\cap	$a \cap b$	a and b
논리합(disjunction)	\cup	$a \cup b$	a or b
동등(equivalence)	\equiv	$a \equiv b$	a is equivalent to b
함축(implication)	\supset	$a \supset b$	a implies b
	\subset	$a \subset b$	b implies a

한정사(Quantifier)

Name	Example	Meaning
universal	$\forall X.P$	For all X, P is true
existential	$\exists X.P$	There exists a value of X such that P is true

Clausal Form

- 문제의 단순화를 위해 명제의 표준 형식을 사용
- *Clausal form*:
 - $B_1 \cup B_2 \cup \dots \cup B_n \subset A_1 \cap A_2 \cap \dots \cap A_m$
 - means if all the As are true, then at least one B is true
- *전제(Antecedent)* : right side
- *결론(Consequent)* : left side

해 도출(Resolution)

- *해 도출(Resolution)* : 추론된 명제가 주어진 명제로부터 계산되도록 해주는 추론 규칙
- *단일화(Unification)* : 명제에서 변수가 있으면 해 도출과정에서 매칭과정이 성공하는 변수의 값을 찾음; 변수의 유용한 값을 결정하는 과정
- *사례화(Instantiation)* : 단일화를 위해 변수에 값을 일시적으로 지정시키는 것
- 변수에 값을 사례화한 후 매칭이 실패하면 역행(*backtrack*)해서 변수에 다른 값을 사례화한다.

정리 증명(Theorem Proving)

- 논리 프로그래밍의 기반
- 명제가 해 도출을 위해 사용될 때, 명제의 제한된 형식만을 사용
- *Horn clause* – 두 개의 형식 사용 가능
 - *Headed* : single atomic proposition on left side
(ex) $\text{likes}(\text{bob}, \text{trout}) \subset \text{likes}(\text{bob}, \text{fish}) \cap \text{fish}(\text{trout})$
 - *Headless* : empty left side (사실 언급하는데 사용)
(ex) $\text{father}(\text{bob}, \text{jake})$
- 대부분의 명제는 혼 클로즈로 나타낼 수 있다.

논리형 프로그래밍의 개관

- 선언적 의미론(Declarative semantics)
 - 각 문장의 의미를 결정하는 간단한 방법이 존재
 - 명령형 언어의 의미론 보다 더 단순
- 프로그래밍이 비절차적
 - 결과가 계산되는 방법을 서술하지 않고 결과의 형식을 기술

Prolog의 기원

- 1970년대 초반, University of Aix-Marseille (Colmerauer & Roussel)
 - Natural language processing
- University of Edinburgh (Kowalski)
 - Automated theorem proving
- Prolog(**PRO**gramming in **LOGic**)

Prolog 인터프리터

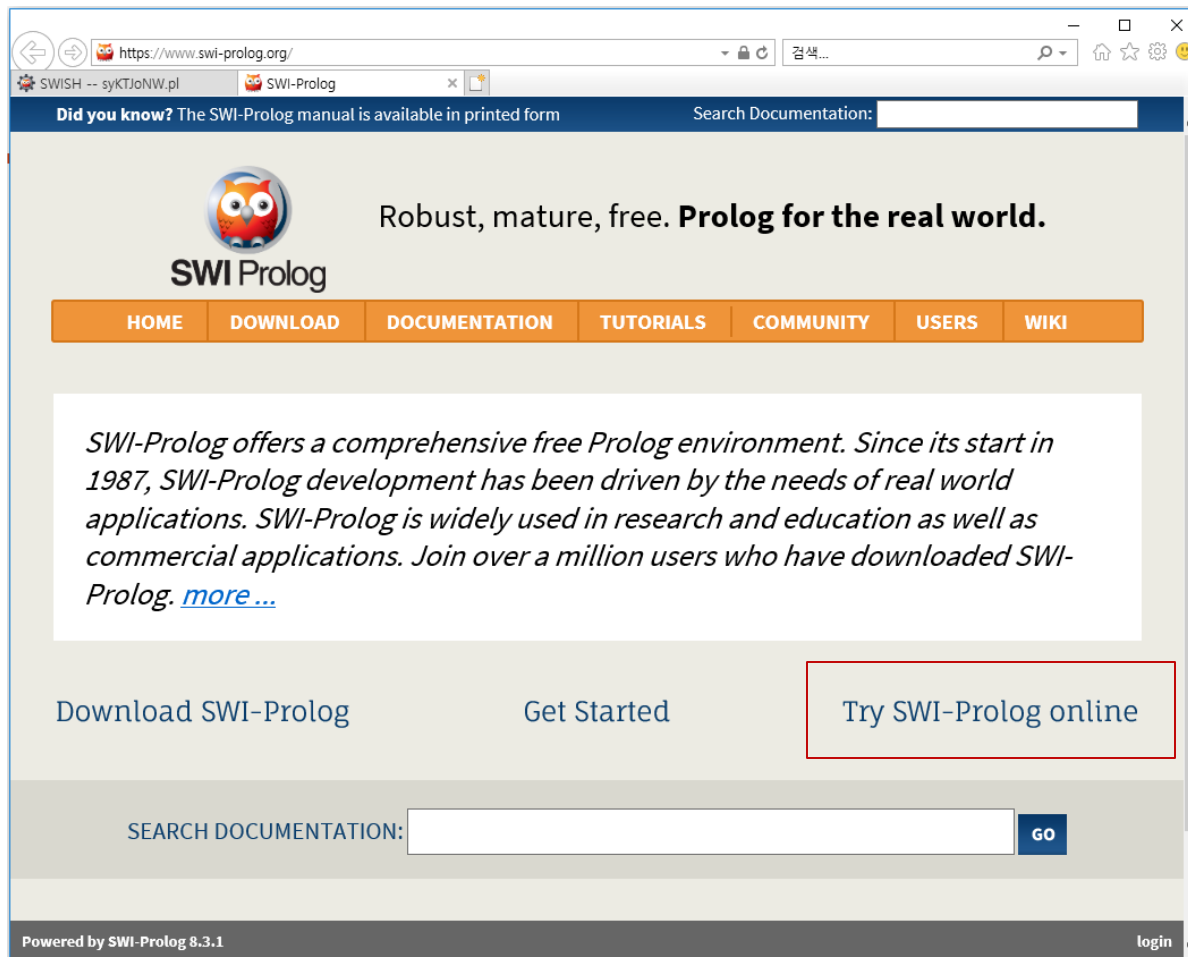
- <http://www.swi-prolog.org/>

[1] 간단한 테스트 : Try SWI-Prolog online

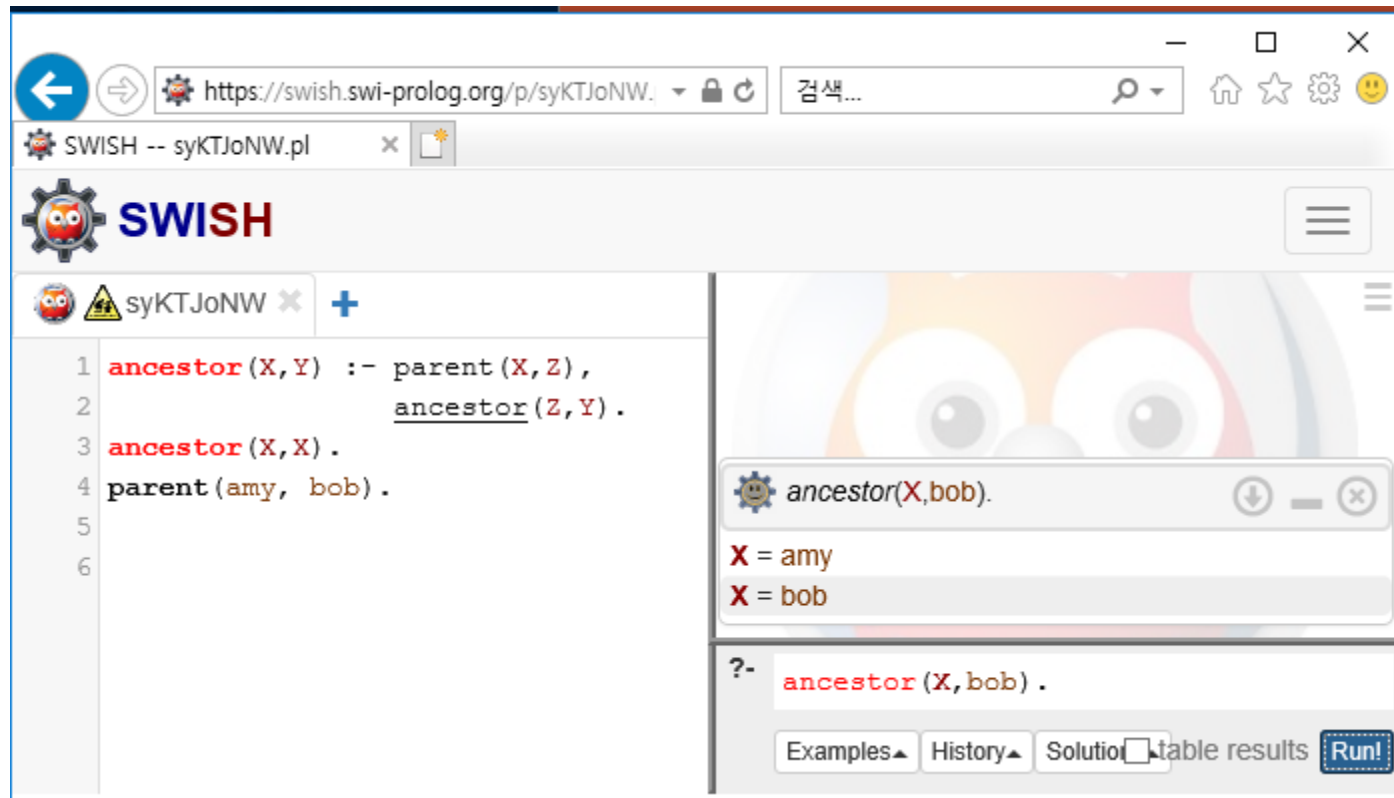
[2] Download : SWI-Prolog 8.2.0

- 사용법
 - ?- 프롬프트에서 질의 입력함
(ex) ?- length([1,2,3,4,5], X).
 ?- append([a,b,c],[d,e,f], X).
 ?- member(c, [a,b,c,d,e,f]).
 - 문장 끝은 반드시 ‘.’
 - 빠져 나오기 : ^D

<http://www.swi-prolog.org/>



[1] 간단한 테스트 : Try SWI-Prolog online



The screenshot shows the SWISH online Prolog environment. The browser address bar displays the URL `https://swish.swi-prolog.org/p/syKTJoNW`. The page title is "SWISH -- syKTJoNW.pl". The main content area shows a Prolog code editor with the following code:


```
1 ancestor(X,Y) :- parent(X,Z),
2                   ancestor(Z,Y).
3 ancestor(X,X).
4 parent(amy, bob).
5
6
```

On the right side, there is a query input field with the text `ancestor(X,bob).` and a "Run!" button. Below the query field, the results are displayed:

```
?- ancestor(X,bob).
X = amy
X = bob
```

The interface includes a sidebar with a "SWISH" logo and a menu icon. The bottom of the page has navigation links: "Examples", "History", "Solution", "table results", and a "Run!" button.

[2] Download : SWI-Prolog 8.2.0



SWI-Prolog downloads

[HOME](#) [DOWNLOAD](#) [DOCUMENTATION](#) [TUTORIALS](#) [COMMUNITY](#) [USERS](#) [WIKI](#)

SWI-Prolog
Sources/building
Docker images
Add-ons
Browse GIT

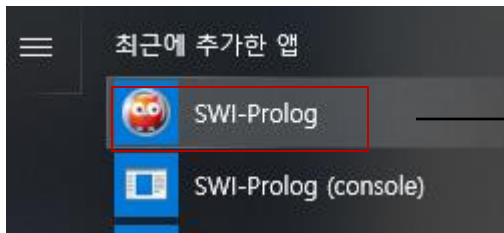
Available versions

The **stable** release is the recommended version for running basic Prolog code without surprises. The **development** version is released roughly every 2-3 weeks and is the recommended version for developers and users of applications such as [SWISH](#) or [ClioPatria](#). Finally, the **GIT** and **daily** versions are for developers that want to contribute or have immediate access to patches. These versions are generally fine, but occasionally suffer from regression.

- [Stable release](#)
- [Development release](#)
- [Daily builds for Windows](#)
- Browse GIT [repository](#)

Read more about

- Available SWI-Prolog [versions](#)
- Information on [Linux packages and building on Linux](#)



실행!

```
SWI-Prolog (Multi-threaded, version 8.2.0)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 32 bits, version 8.2.0)
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).

?- █
```

사용법

- File->New 메뉴에서 file 이름을 test로 입력
- 코드 입력(slide 18의 왼쪽 창에 입력한 내용) 후
Save buffer 메뉴 선택 후 빠져나옴
- File->consult 메뉴에서 test.pl 선택 후 열기
- ?- 프롬프트에서 질의 입력함
(ex) ?- ancestor(X, bob).

Prolog의 기본 표기법

- “if”의 의미로 “:-” 사용
- 변수 : 대문자 또는 “_”로 시작
- 상수, 술어이름, 함수이름 : 소문자로 시작
- **AND**연결자(,)와 **OR**연결자(;)
- 항(Term) : 술어 또는 함수
 - parent(X,Z).
 - tree(3, tree(5, null, null), tree(1, null, null))
- 리스트 : 콤마와 대괄호 사용
 - [x, y, z] [x|[y, z]] [H|T] []

Prolog의 수행

(1) 프로그램이 입력되어 DB에 저장

규칙: `ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).`
`ancestor(X, X).`

사실: `parent(amy, bob).`

(2) 질의 입력을 위한 프롬프트

?-

(3) 질의를 입력하면 답을 구함

?- `ancestor(X, bob).`

`X=amy;`

`X=bob`

사실(fact)

- 객체 또는 객체들간의 객관적인 관계 선언
- **Headless Horn clauses**
- 일반형식 : **predicate(arg-1, arg-2, ..., arg-n).**
- 괄호 안의 객체들은 순서에 따라 일관성 있게 의미를 부여

(ex)

```
female(shelley) .
```

```
male(bill) .
```

```
father(bill, jake) .
```

규칙(Rule)

- 객체들 간의 관계를 설명하는 규칙
- Headed Horn clause
- 우측 : condition (*if* part)
 - 단일항이거나 ‘,’(논리곱) 이용
- 좌측 : action (*then* part)
 - 단일항
- 일반형식
action :- conditions.

Example Rules

```
ancestor(mary, shelley) :- mother(mary, shelley) .
```

- Can use variables (*universal objects*) to generalize meaning:

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

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

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

목적문(질의문)

- 정리증명을 위해, 정리는 시스템이 증명하기 위한 명제의 형식으로 되어 있다. 이러한 명제가 목적문(*goal statement*) 임

1. Headless Horn과 동일한 형식의 목적문

- True/False 답변

(ex) ?- man(fred) .

2. 변수를 가진 목적문

(ex) ?- father(X, mike) .

X=bill

(ex) ?- mother(X, bill) .

X=marry

간단한 산술연산

- **Prolog**는 정수 변수와 정수 산술 연산을 지원
- **is** 연산자 : 오른쪽 피연산자는 산술표현식을, 왼쪽 피연산자는 변수를 취하게 함

(ex) `?- A is B / 17 + C.`

- `write()`는 출력을 위한 내장 술어

(ex) `?- X is 3+5, write(X).`

`X=8`

- 다음 `=`은 단일화 연산자

(ex) `?- 3+5=4+3.`

`false`

단일화(Unification)

- 두 항을 같게 만드는 과정(패턴 매칭 사용)
- 변수의 경우는 값을 바인딩("실체화" 라고 함)

(ex) `?- me=me.`

`true.`

`?- me=X.`

`X=me.`

`?- f(a, X)=f(Y, b).`

`X=b,`

`Y=a.`

`?- f(X)=g(X).`

`false.`

Prolog의 추론과정

- Queries are called goals
- If a goal is a compound proposition, each of the facts is a subgoal
- To prove a goal is true, must find a chain of inference rules and/or facts. For goal Q:

$P_2 \text{ :- } P_1$

$P_3 \text{ :- } P_2$

...

$Q \text{ :- } P_n$

- Process of proving a subgoal called matching, satisfying, or resolution

Approaches

- *Matching* is the process of proving a proposition
- Proving a subgoal is called *satisfying* the subgoal
- *Bottom-up resolution, 전향 체인(forward chaining)*
 - Begin with facts and rules of database and attempt to find sequence that leads to goal
 - Works well with a large set of possibly correct answers
- *Top-down resolution, 후향 체인(backward chaining)*
 - Begin with goal and attempt to find sequence that leads to set of facts in database
 - Works well with a small set of possibly correct answers
- Prolog는 backward chaining을 사용하여 구현

Approaches

(Ex) goal : man(bob).

-- DB의 내용 --

father(bob).

man(X) :- father(X).

- 전향 체인(*forward chaining*)

- 첫 번째 명제를 탐색; X를 bob으로 사례화; 첫 번째 명제를 두 번째 규칙의 우변 (father(X))과 매칭 후 두 번째 명제의 좌변을 목적에 매칭

- 후향 체인(*backward chaining*)

- X를 bob으로 사례화; 목적을 두 번째 명제의 좌변(man(X))과 매칭; 두 번째 명제의 우변 (father(X))은 첫 번째 명제와 매칭

Subgoal Strategies

- When goal has more than one subgoal, can use either
 - 깊이-우선 탐색(Depth-first search): find a complete proof for the first subgoal before working on others
 - 너비-우선 탐색(Breadth-first search): work on all subgoals in parallel
- Prolog는 깊이-우선탐색을 사용
 - Can be done with fewer computer resources

역행(Backtracking)

- With a goal with multiple subgoals, if fail to show truth of one of subgoals, reconsider previous subgoal to find an alternative solution: *backtracking*
- Begin search where previous search left off
- Can take lots of time and space because may find all possible proofs to every subgoal

Simple Arithmetic

- Prolog supports integer variables and integer arithmetic
- **is** operator: takes an arithmetic expression as right operand and variable as left operand

`A is B / 17 + C`

- Not the same as an assignment statement!
 - The following is illegal:
`Sum is Sum + Number.`

자동차 경주 트랙의 예

```
speed(ford,100).
speed(chevy,105).
speed(dodge,95).
speed(volvo,80).
time(ford,20).
time(chevy,21).
time(dodge,24).
time(volvo,24).
distance(X,Y) :-    speed(X,Speed),
                    time(X,Time),
                    Y is Speed * Time.
```

A query: distance(chevy, Chevy_Distance).

Trace

- 각 단계에서 수행한 변수의 사례화를 디스플레이하는 내장 구조
- *Tracing model* of execution – four events:
 - *Call* (beginning of attempt to satisfy goal)
 - *Exit* (when a goal has been satisfied)
 - *Redo* (when backtrack occurs)
 - *Fail* (when goal fails)

자동차 경주 트랙의 예

trace.

distance(chevy, Chevy_Distance).

(1) 1 Call: distance(chevy, _0)?

(2) 2 Call: speed(chevy, _5)?

(2) 2 Exit: speed(chevy, 105)

(3) 2 Call: time(chevy, _6)?

(3) 2 Exit: time(chevy, 21)

(4) 2 Call: _0 is 105*21?

(4) 2 Exit: 2205 is 105*21

(1) 1 Exit: distance(chevy, 2205)

Chevy_distance = 2205

```
speed(ford,100).
speed(chevy,105).
speed(dodge,95).
speed(volvo,80).
time(ford,20).
time(chevy,21).
time(dodge,24).
time(volvo,24).
distance(X,Y) :- speed(X,Speed),
                  time(X,Time),
                  Y is Speed * Time.
```

A query: distance(chevy, Chevy_Distance).

Example

```
likes(jake, chocolate).  
likes(jake, apricots).  
likes(darcie, licorice).  
likes(darcie, apricots).
```

trace.

```
likes(jake, X), likes(darcie, X).
```

```
(1) 1 Call: likes(jake, _0)?
```

```
(1) 1 Exit: likes(jake, chocolate)
```

```
(2) 1 Call: likes(darcie, chocolate)?
```

```
(2) 1 Fail: likes(darcie, chocolate)
```

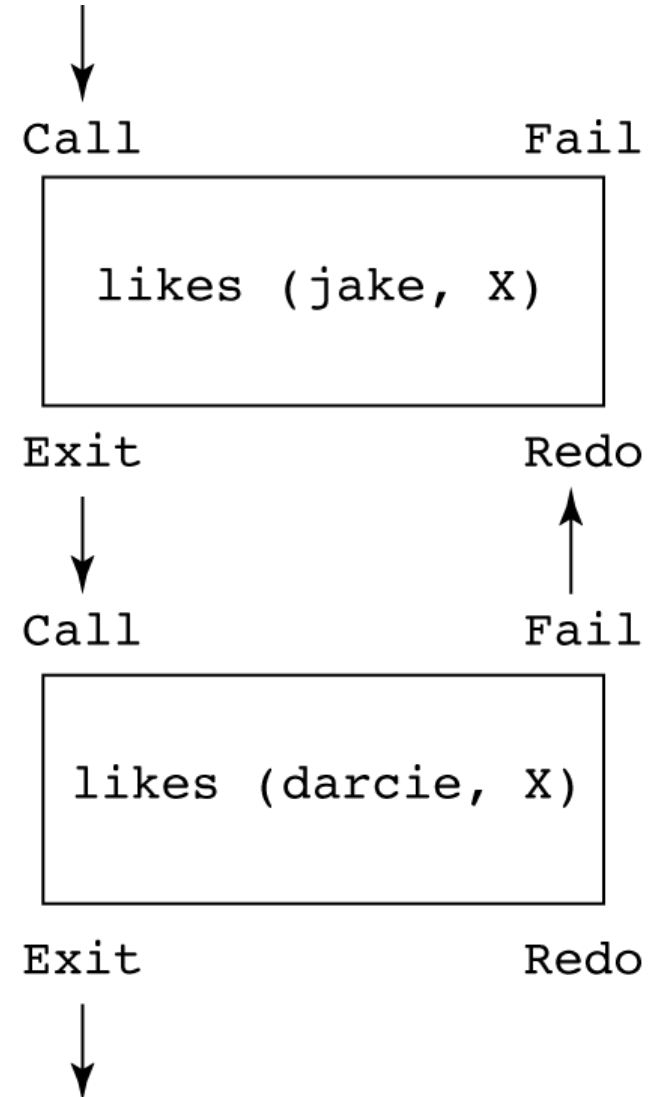
```
(1) 1 Redo: likes(jake, _0)?
```

```
(1) 1 Exit: likes(jake, apricots)
```

```
(3) 1 Call: likes(darcie, apricots)?
```

```
(3) 1 Exit: likes(darcie, apricots)
```

```
X = apricots
```



List Structures

- Other basic data structure (besides atomic propositions we have already seen): list
- *List* is a sequence of any number of elements
- Elements can be atoms, atomic propositions, or other terms (including other lists)

`[apple, prune, grape, kumquat]`

`[]` *(empty list)*

`[X | Y]` *(head X and tail Y)*

Append Example

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

```
append([Head | List_1], List_2, [Head | List_3]) :-  
    append (List_1, List_2, List_3).
```

```
trace.
```

```
append([bob, jo],[jake,darcie], Family).
```

```
(1) 1 Call: append([bob, jo],[jake,darcie], _10)?
```

```
(2) 2 Call: append([jo],[jake,darcie], _18)?
```

```
(3) 3 Call: append([], [jake,darcie], _25)?
```

```
(3) 3 Exit: append([], [jake,darcie], [jake,darcie])
```

```
(2) 2 Exit: append([jo],[jake,darcie], [jo,jake,darcie])
```

```
(1) 1 Exit: append([bob,jo],[jake,darcie], [bob,jo,jake,darcie])
```

```
Family = [bob,jo,jake,darcie]
```

?– append(X, Y, [a, b, c]).

X=[],

Y=[a, b, c];

X=[a],

Y=[b, c];

X=[a, b],

Y=[c];

X=[a, b, c],

Y=[];

Prolog의 결점

- 해 도출 순서 제어
 - 순수 논리형 프로그래밍 환경에서 매칭의 순서는 비결정적이며 모든 매칭은 병렬적으로 시도될 수 있음
 - Prolog는 사용자에게 DB와 subgoal의 순서제어를 허용
- 닫힌 세계 가정(closed-world assumption)
 - The only knowledge is what is in the database
- 부정 문제(negation problem)
 - Anything not stated in the database is assumed to be false
- 본질적인 한계(Intrinsic limitation)
 - It is easy to state a sort process in logic, but difficult to actually do—it doesn't know how to sort

Applications of Logic Programming

- Relational database management systems
- Expert systems
- Natural language processing

Summary

- Symbolic logic provides basis for logic programming
- Logic programs should be nonprocedural
- Prolog statements are facts, rules, or goals
- Resolution is the primary activity of a Prolog interpreter
- Although there are a number of drawbacks with the current state of logic programming it has been used in a number of areas