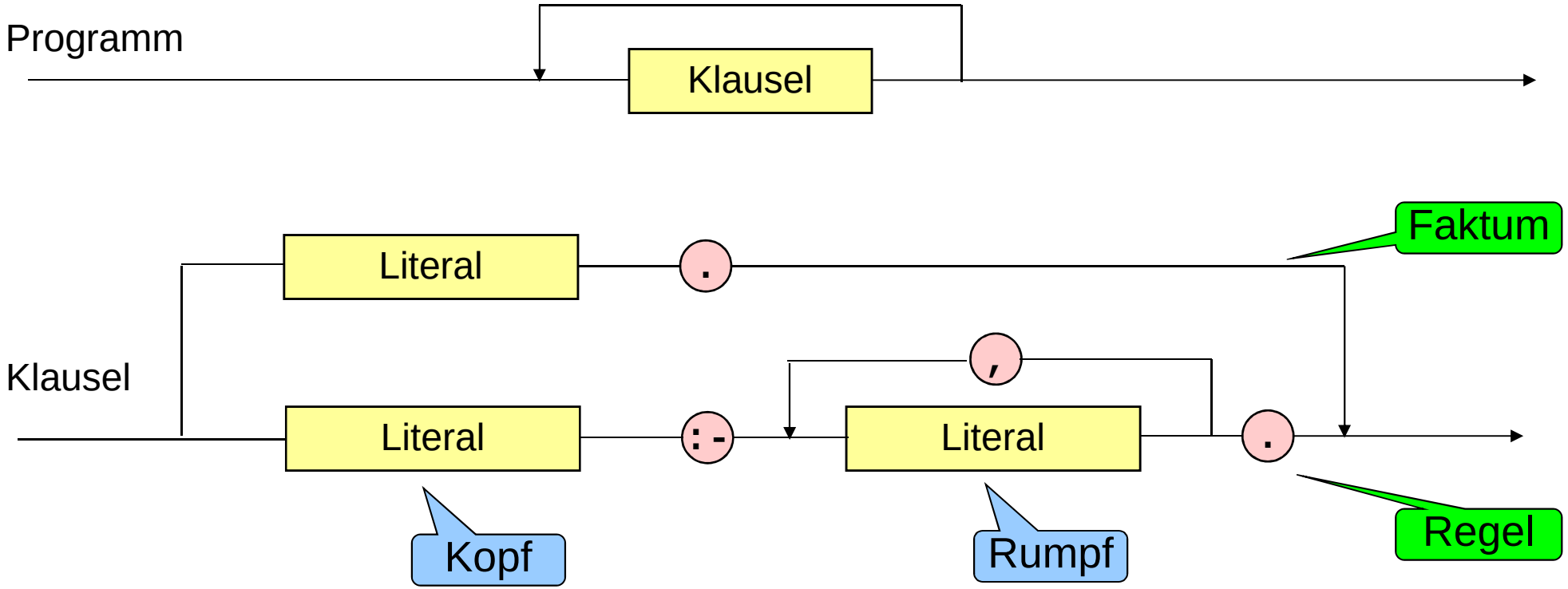
IV. Logische Programmierung

- 1. Grundkonzepte der logischen Programmierung
- 2. Syntax von Prolog
- 3. Rechnen in Prolog

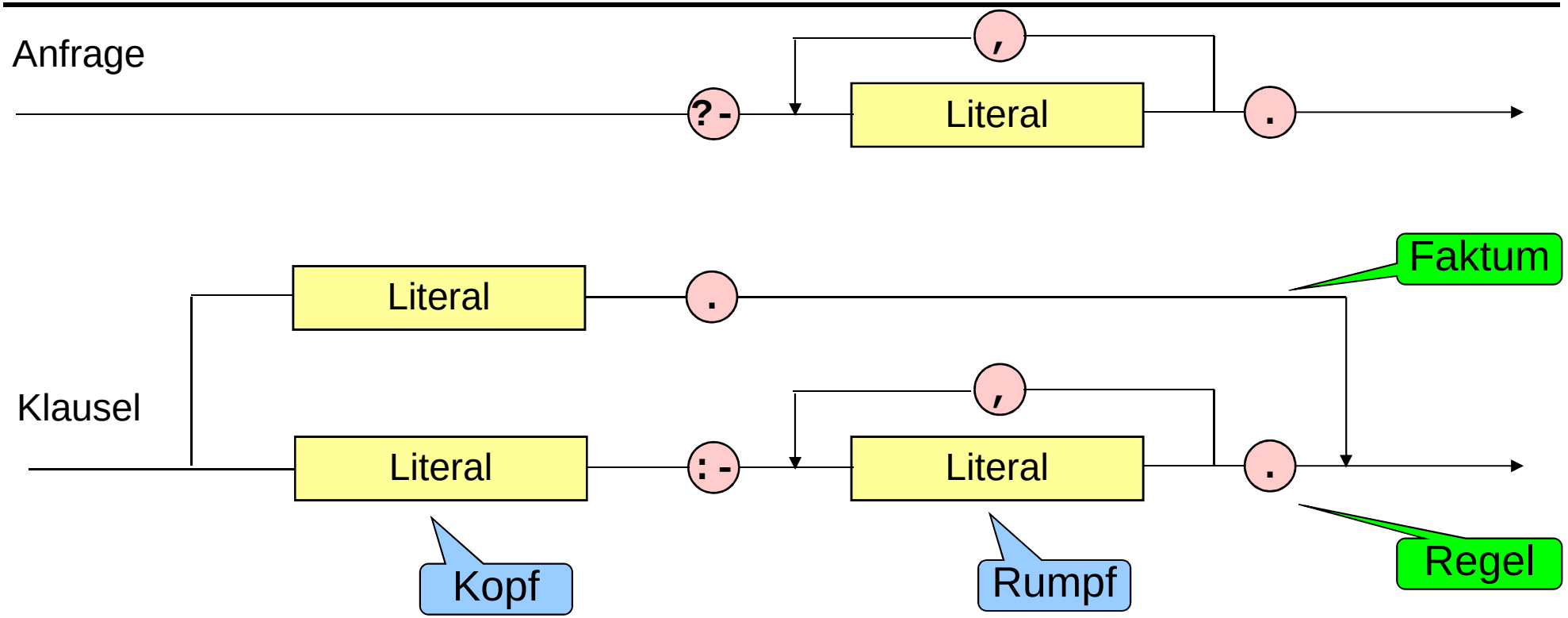
Syntax




```
weiblich(monika).                    maennlich(werner).
...                                  ...
verheiratet(klaus, susanne).         mutterVon(susanne, dominique).

vaterVon(V,K)    :- verheiratet(V,F), mutterVon(F,K).
```

Syntax

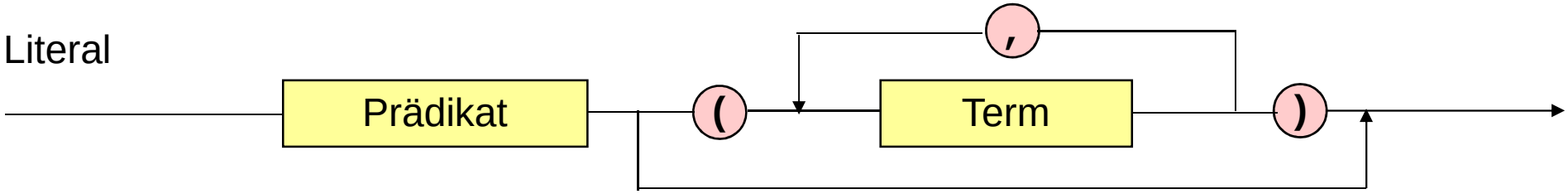



```
weiblich(monika).                    maennlich(werner).
...                                  ...
verheiratet(klaus, susanne).         mutterVon(susanne, dominique).

vaterVon(V,K)    :- verheiratet(V,F), mutterVon(F,K).
```

```
?- verheiratet(gerd,F), mutterVon(F,susanne).
```

Syntax



Prädikate: **weiblich, verheiratet, vaterVon,...**

Literale: **weiblich(monika), verheiratet(gerd,F), ...**

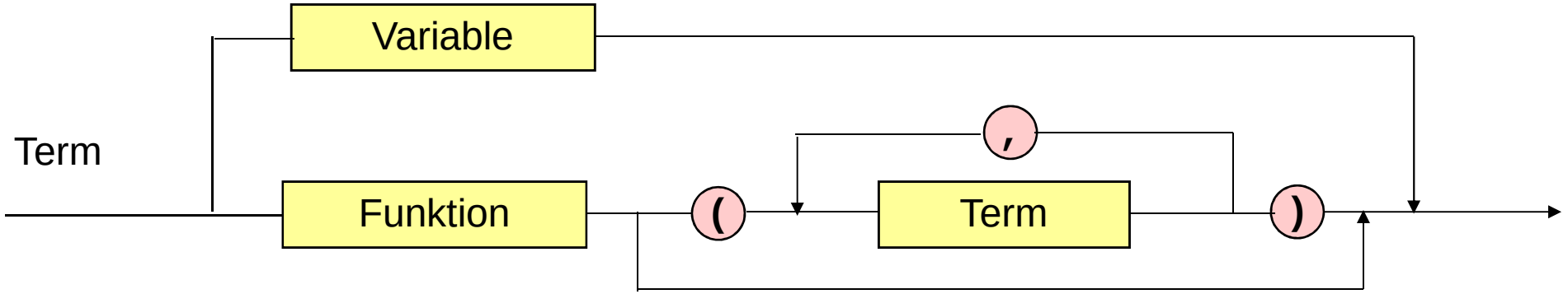
Terme: **monika, werner, V, F, ...**

```
weiblich(monika).                    maennlich(werner).
...                                  ...
verheiratet(klaus, susanne).         mutterVon(susanne, dominique).

vaterVon(V,K)    :- verheiratet(V,F), mutterVon(F,K).
```

```
?- verheiratet(gerd,F), mutterVon(F,susanne).
```

Syntax



Terme: **monika, werner, V, F, ...**

Variable: **V, F, _,...**

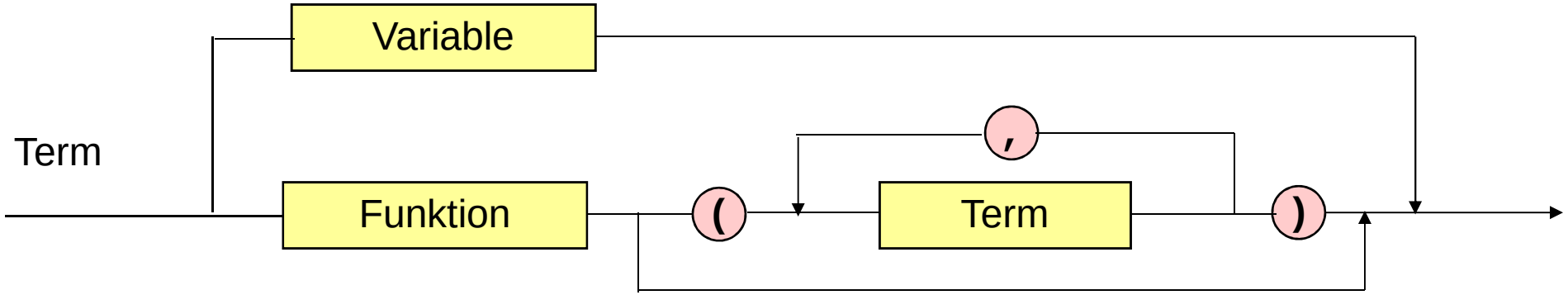
Funktionen (Konstanten): **monika, werner, ...**

```
weiblich(monika).                    maennlich(werner).
...                                  ...
verheiratet(klaus, susanne).         mutterVon(susanne, dominique).

vaterVon(V,K)    :- verheiratet(V,F), mutterVon(F,K).
```

```
?- verheiratet(gerd,F), mutterVon(F,susanne).
```

Syntax



Terme: **monika, werner, V, F, ...**

Variable: **V, F, _,...**

Funktionen (Konstanten): **monika, werner, ...**

Funktionen (keine Konstanten): **datum, ...**

```
geboren(monika, datum(25,7,1972)).
geboren(werner, datum(12,7,1969)).
```

```
?- geboren(monika,X).

X = datum(25,7,1972)
```

```
?- geboren(X, datum(_,7,_)).

X = monika ; X = werner
```

Datenstrukturen in Prolog: Zahlen

Datenstruktur für natürliche Zahlen:

Terme aus den Funktionssymbolen **`zero`** und **`succ`**

```
A.   Die Addition von X und zero ist X.

B.   Die Addition von X und succ(Y) ist succ(Z),
     falls die Addition von X und Y den Wert Z ergibt.
```

```
add(X, zero, X).
add(X, succ(Y), succ(Z)) :- add(X, Y, Z).
```

```
?- add(succ(zero),succ(zero),U).
U = succ(succ(zero))

?- add(succ(zero),V,succ(succ(zero))).
V = succ(zero)
```

```
?- add(succ(_),_,zero).
false
```

Datenstrukturen in Prolog: Zahlen

A. Die Multiplikation von X und zero ist zero.

B. Die Multiplikation von X und succ(Y) ist Z,
falls die Multiplikation von X und Y den Wert U hat
und die Addition von X und U den Wert Z ergibt.

```
mult(X, zero, zero).
mult(X, succ(Y), Z) :- mult(X, Y, U), add(X, U, Z).
```

```
add(X, zero, X).
add(X, succ(Y), succ(Z)) :- add(X, Y, Z).
```

```
?- mult(succ(zero),succ(zero),U).
U = succ(zero)
```

```
?- mult(succ(_),Y,zero).
Y = zero
```

```
?- mult(succ(zero),V,succ(succ(zero))).
V = succ(succ(zero))
```

Datenstrukturen in Prolog: Listen

Datenstruktur für Listen: Funktionssymbole **nil** und **cons**

A. Die Länge der leeren Liste nil ist zero.

B. Die Länge der Liste cons(Kopf, Rest) ist succ(N), falls N die Länge der Liste Rest ist.

```
len(nil, zero).
len(cons(Kopf, Rest), succ(N)) :- len(Rest, N).
```

```
?- len(cons(zero, cons(succ(zero), nil)), U).
U = succ(succ(zero))
```

```
?- len(L,succ(succ(zero))).
L = cons(A, cons(B, nil))
```

Vordefinierte Listen

Datenstruktur für Listen: Funktionssymbole **`nil`** und **`cons`**

Vordefinierte Listen: Funktionssymbole **`[]`** und **`.`**

```
leng([], zero).
leng([Kopf | Rest], succ(N)) :- leng(Rest, N).
```

```
len(nil, zero).
len(cons(Kopf, Rest), succ(N)) :- len(Rest, N).
```

```
?- len(cons(...), U).
U = succ(succ(zero))
```

```
?- leng([zero, succ(zero)], U).
U = succ(succ(zero))
```

```
?- len(L,succ(succ(zero))).
L = cons(A, cons(B, nil))
```

```
?- leng(L,succ(succ(zero))).
L = [A, B]
```