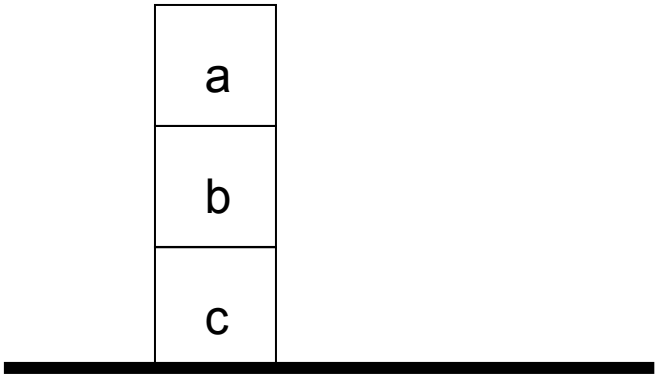
Logische Sprachen

Anwendung: Robotik, Erstellung von Roboterbewegungsplänen



auf(a,b).

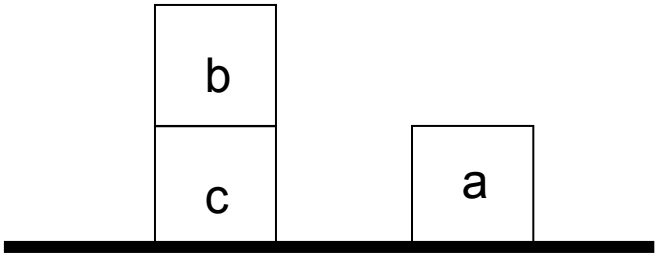
auf(b,c).

auf(c,tisch).

Logische Sprachen

Bewegung: stelle a auf den Tisch

(Tisch hat beliebig Platz, Würfel werden einzeln übereinander positioniert)



Wunsch-Wissensbasis danach:

```
auf(a,tisch).
auf(b,c).
auf(c,tisch).
bewegt(a,b,tisch).
```

Logische Sprachen

Regel für Bewegung von Würfel A auf B:

```
stelle_auf(A,B) :-
    A =\= tisch,
    A =\= B,
    auf(A,X),
    frei(A),            %auf A darf nichts stehen
    frei(B),            %auf B darf nichts stehen
    retract(auf(A,X)),
    assert(auf(A,B)),
    assert(bewegt(A,X,B)).

frei(tisch).
frei(X) :- not(auf(_,X)).
```

Logische Sprachen

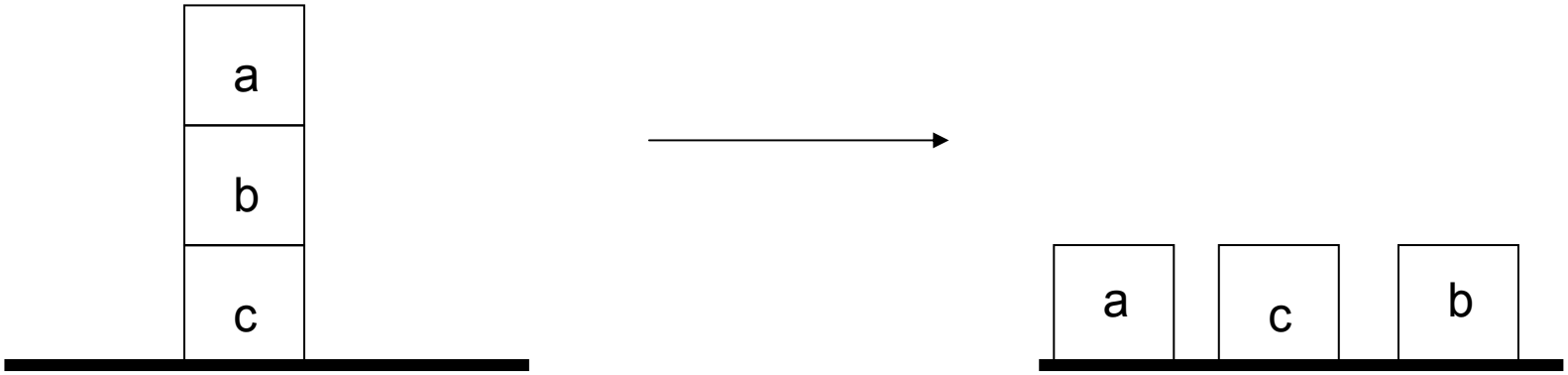
```
?-assert(auf(c,tisch)).
?-assert(auf(b,c)).
?-assert(auf(a,tisch)).
?-stelle_auf(b,a).          %erzeugt in WB:

?-listing(auf).
auf(c,tisch).
auf(a,tisch).
auf(b,a).

?-listing(bewegt).
bewegt(b,c,a).
```

Wunsch: auch Bewegungsplan für Block, der zwischen anderen steht

Logische Sprachen




```
auf(a,b).
auf(b,c).
auf(c,tisch).
```

→

```
auf(a,tisch).
auf(b,tisch).
auf(c,tisch).
bewegt(a,b,tisch).
bewegt(b,c,tisch).
```

→ rekursiv Blöcke wegräumen, bis Block frei ist.

Logische Sprachen

```
stelle_auf(A,B) :- auf(A,B).
stelle_auf(A,B) :- not(auf(A,B)),
    A =\= tisch,
    A =\= B,
    auf(A,X),
    frei(A),    %A wird freigeräumt falls belegt
    frei(B),    %B wird freigeräumt falls belegt
    retract(auf(A,X)),
    assert(auf(A,B)),
    assert(bewegt(A,X,B)).

frei(tisch).
frei(X) :- not(auf(_,X)).
frei(X) :- X\=tisch, auf(Y,X),frei(Y),%hier Rekursion
           retract(auf(Y,X)),assert(auf(Y,tisch)),
           assert(bewegt(Y,X,tisch)).
```

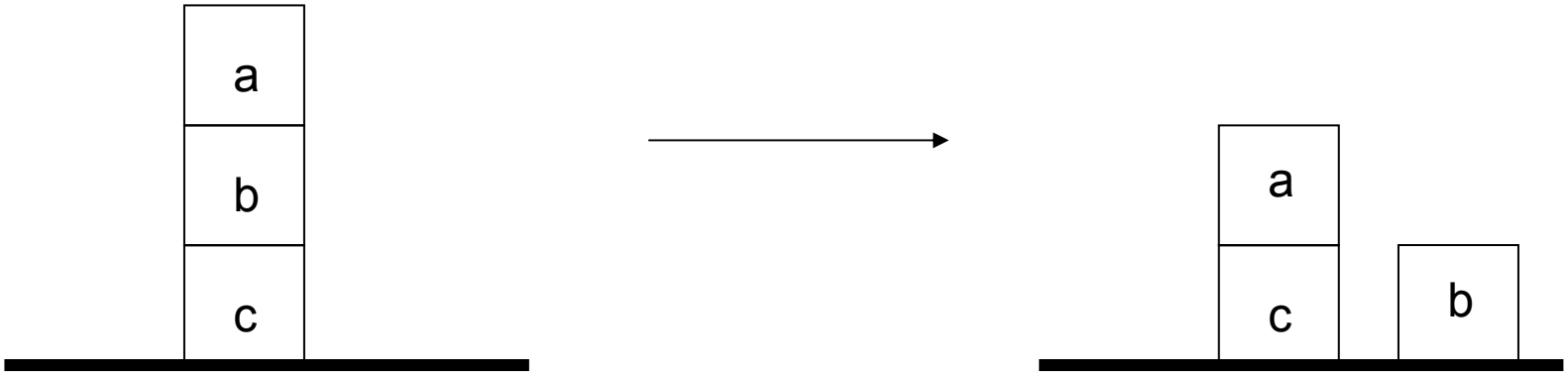
Logische Sprachen

```
?-stelle_auf(b,tisch),stelle_auf(a,c).

?-listing(auf).
auf(c,tisch).
auf(b,tisch).
auf(a,c).

?-listing(bewegt).
bewegt(a,b,tisch).
bewegt(b,c,tisch).
bewegt(a,tisch,c).
```

Logische Sprachen




```
?-stelle_auf(a,c).
```

Weitere Erweitung notwendig, wenn

- Block zum Freiräumen eines anderen bewegt wurde
- selbst danach positioniert werden soll

Logische Sprachen

```
stelle_auf(A,B) :- auf(A,B).
stelle_auf(A,B) :- not(auf(A,B)),A\=tisch,A\=B,
    auf(A,X),frei(A),frei(B),
    myretract_auf(A,X,XN),
    assert(auf(A,B)),assert(bewegt(A,XN,B)).

% if auf(A,B) lösche dies, else lösche auf(A,X)
myretract_auf(A,B,B) :-auf(A,B),!,retract(auf(A,B)).
myretract_auf(A,B,X) :-auf(A,X),X\=B,retract(auf(A,X)).

frei(tisch).
frei(X) :- not(auf(_,X)).
frei(A) :- A\=tisch,
           auf(X,A), frei(X),
           myretract_auf(X,A,AN), assert(auf(X,tisch)),
           assert(bewegt(X,AN,tisch)).
```

Logische Sprachen

Kürzer:

```
stelle_auf(A,B) :- not(auf(A,B)),A\=tisch,A\=B,
    auf(A,X),frei(A),frei(B),
    retract(auf(A,Y)),
    assert(auf(A,B)),assert(bewegt(A,Y,B)).
...

frei(tisch).
frei(X) :- not(auf(_,X)).
frei(A) :- A\=tisch,
           auf(X,A), frei(X),
           retract(auf(X,Y)), assert(auf(X,tisch)),
           assert(bewegt(X,Y,tisch)).
```

Logische Sprachen

Noch kürzer:

```
stelle_auf(A,B) :- not(auf(A,B)),A\=tisch,A\=B,
    frei(A),frei(B), %auf(A,X) kann ganz entfallen
    retract(auf(A,X)),
    assert(auf(A,B)),assert(bewegt(A,X,B)).
...
frei(tisch).
frei(X) :- not(auf(_,X)).
frei(A) :- A\=tisch,
           auf(X,A), frei(X),
           retract(auf(X,Y)), assert(auf(X,tisch)),
           assert(bewegt(X,Y,tisch)).
```