Prolog lecture 4

Go to:

http://etc.ch/8WDC

Or scan the barcode

Today's discussion

Videos:

- Generate and Test
- Symbolic

There are 119 slides in today's deck

Q: Regarding cut, If we have rule :- generate, !, test - would this evaluate to false (& thus miss solutions) if test is not true for the first generated solution?

A: Yes. But we're not talking about cut today...

Implement take(List,E,R).

```
% take(List,E,R) succeeds if you can remove element E from List
% and R is left.

Vote when finished!
```

http://etc.ch/8WDC

DirectPoll

Implement take(List,E,R).

```
% take(List,E,R) succeeds if you can remove element E from List
% and R is left.

take([H|T],H,T).          % if you take H from a list containing
                          % [H|T] you are left with T

...
```

Implement take(List,E,R).

```
% take(List,E,R) succeeds if you can remove element E from List
% and R is left.

take([H|T],H,T).            % if you take H from a list containing
                            % [H|T] you are left with T

take([H|T],E,[H|R]) :- take(T,E,R). % you can take E from some
                                    % list with head H and tail T
                                    % leaving a list with head H
                                    % and tail R if you can take E
                                    % from T leaving R.
```

Q: How do we know which way is the backtracking way and which is the normal way? It is probably simple in cases of take, etc. but what about more complex cases?

A: Assuming this refers to the questions about using things 'backwards'...

There's a Prolog convention for comments

```
% take(+List,-H,-T)
% remove an element H from List, unify T with what's left

take([H|T],H,T).                    % if you take H from a list containing [H|T] you are left with T

take([H|T],E,[H|R]) :- take(T,E,R).  % you can take E from some list with head H and tail T leaving a list with head H and tail R if you can take E from T leaving R.
```

This are called 'modes'

++	The argument is ground (no variables anywhere).
-	The argument is an 'output' argument.
?	Anything.

I've missed some of the classes out in the interests of time. See the full list here: 'Type, mode and determinism declaration headers'
http://www.swi-prolog.org/pldoc/man?section=modes

If there are no mode comments you need to work it out for yourself

Convention is that inputs (ground terms) come first and output (unbound terms) come last.

But this is only a convention!

What happens if we use take(+List,-E,-R) backwards?

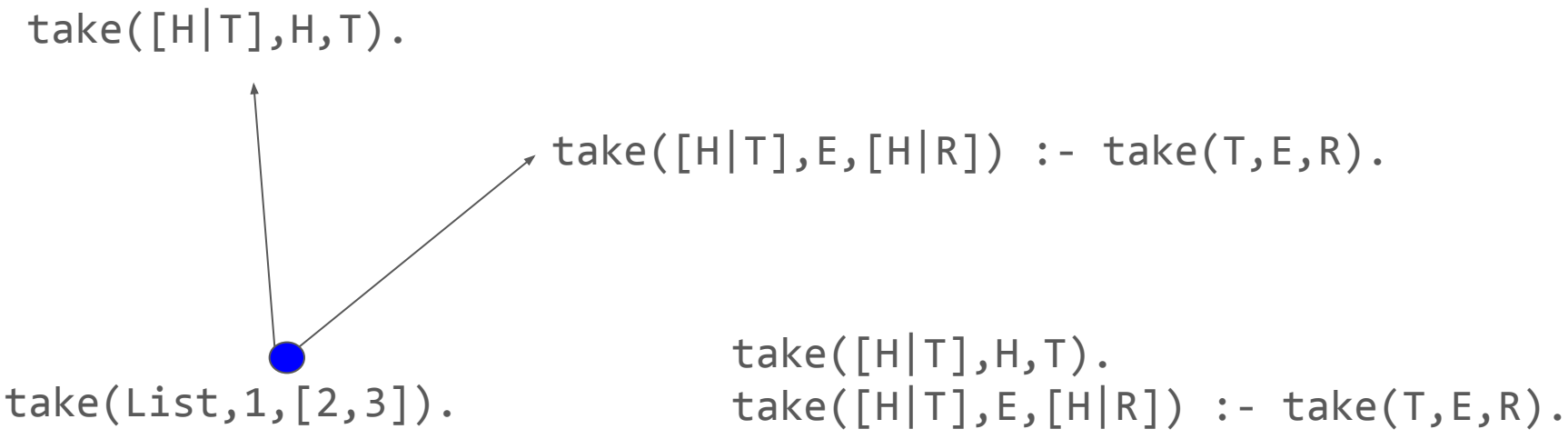
```
take(List,1,[2,3]).
```

```
take(List,1,[2,3]).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

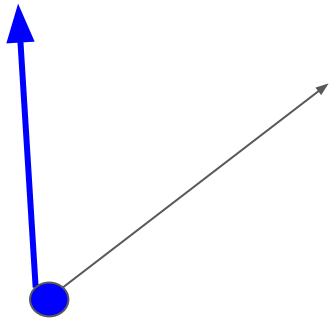
DirectPoll

Q: What's the blue dot?



[H|T] = List
H = 1
T = [2,3]

```
take([H|T],H,T).
```



```
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take(List,1,[2,3]).
```

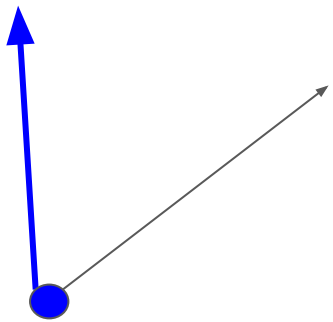
```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

[H|T] = List

H = 1

T = [2,3]

```
take([H|T],H,T).
```



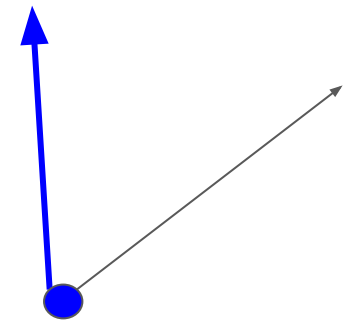
```
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take(List,1,[2,3]).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

[1|T] = List
T = [2,3]

```
take([1|T],1,T).
```



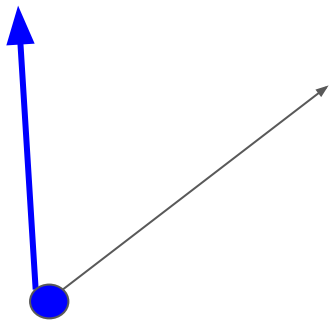
```
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take(List,1,[2,3]).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

[1|T] = List
T = [2,3]

```
take([1|T],1,T).
```



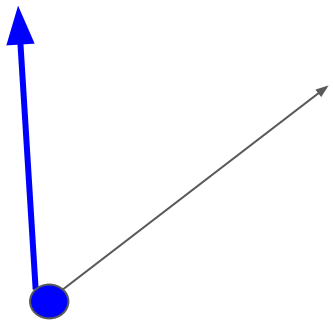
```
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take(List,1,[2,3]).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

[1,2,3] = List

```
take([1,2,3],1,[2,3]).
```



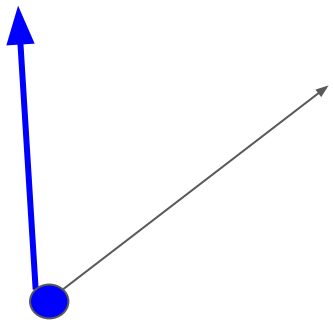
```
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take(List,1,[2,3]).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

[1,2,3] = List

```
take([1,2,3],1,[2,3]).
```

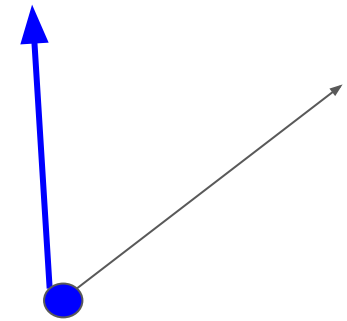


```
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take(List,1,[2,3]).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take([1,2,3],1,[2,3]).
```

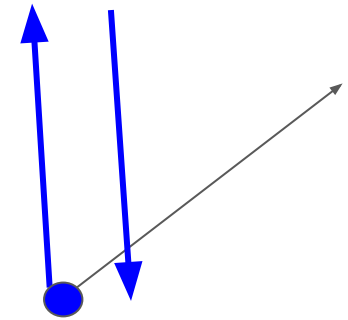


```
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take([1,2,3],1,[2,3]).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

take([1,2,3],1,[2,3]).

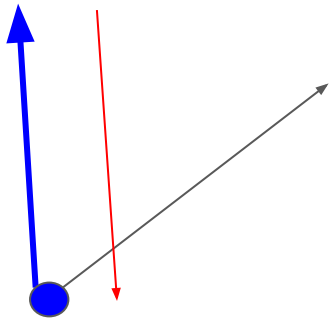


take([H|T],E,[H|R]) :- take(T,E,R).

take([1,2,3],1,[2,3]).

take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).

```
take([1,2,3],1,[2,3]).
```

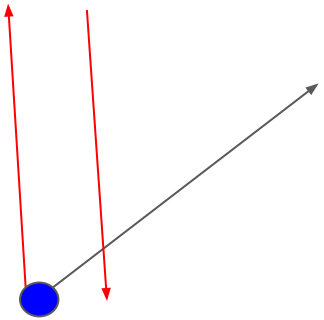


```
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take([1,2,3],1,[2,3]).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take([H|T],H,T).
```



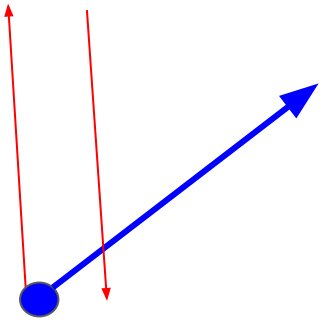
```
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take(List,1,[2,3]).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

[H|T] = List
E = 1
[H|R] = [2,3]

```
take([H|T],H,T).
```



```
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take(List,1,[2,3]).
```

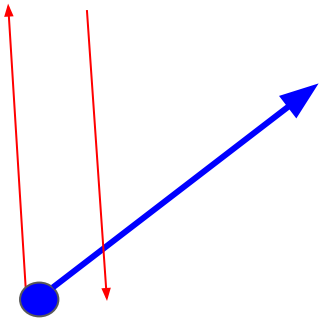
```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

[H|T] = List

E = 1

[H|R] = [2,3]

```
take([H|T],H,T).
```



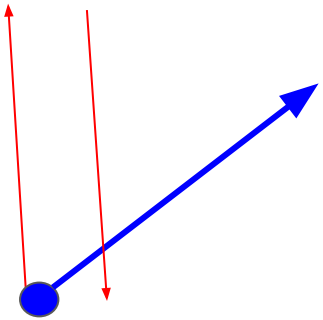
```
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take(List,1,[2,3]).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

[H|T] = List
[H|R] = [2,3]

```
take([H|T],H,T).
```



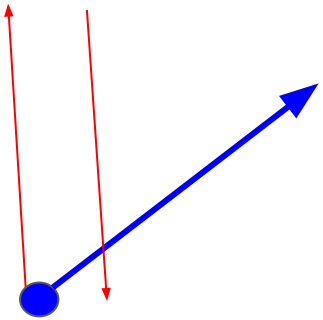
```
take([H|T],1,[H|R]) :- take(T,1,R).
```

```
take(List,1,[2,3]).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

[H|T] = List
[H|R] = [2,3]

```
take([H|T],H,T).
```



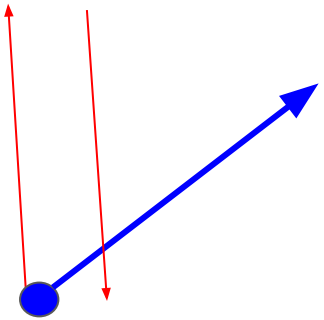
```
take([H|T],1,[H|R]) :- take(T,1,R).
```

```
take(List,1,[2,3]).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

[2|T] = List
[2|R] = [2,3]

```
take([H|T],H,T).
```



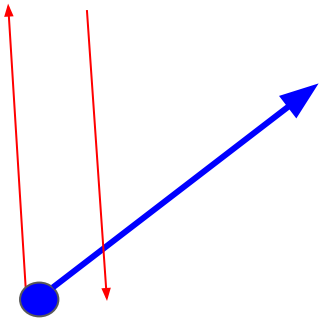
```
take([2|T],1,[2|R]) :- take(T,1,R).
```

```
take(List,1,[2,3]).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

[2|T] = List
[2|R] = [2,3]

```
take([H|T],H,T).
```



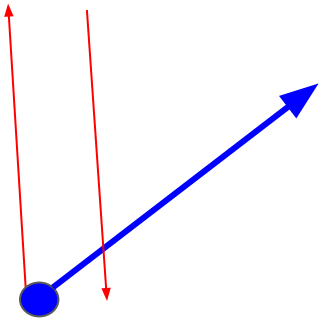
```
take([2|T],1,[2|R]) :- take(T,1,R).
```

```
take(List,1,[2,3]).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take([H|T],H,T).
```

[2|T] = List



```
take([2|T],1,[2,3]) :- take(T,1,[3]).
```

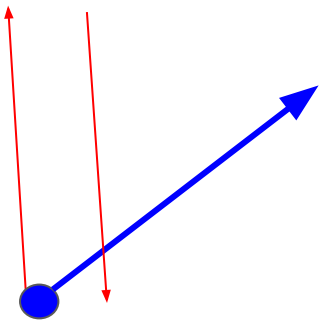
```
take(List,1,[2,3]).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take([H|T],H,T).
```

[2|T] = List

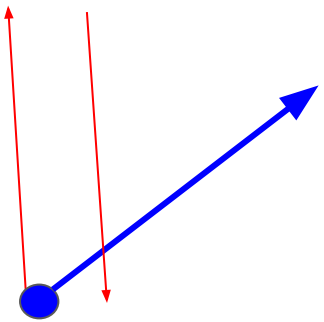
```
take([2|T],1,[2,3]) :- take(T,1,[3]).
```



```
take(List,1,[2,3]).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

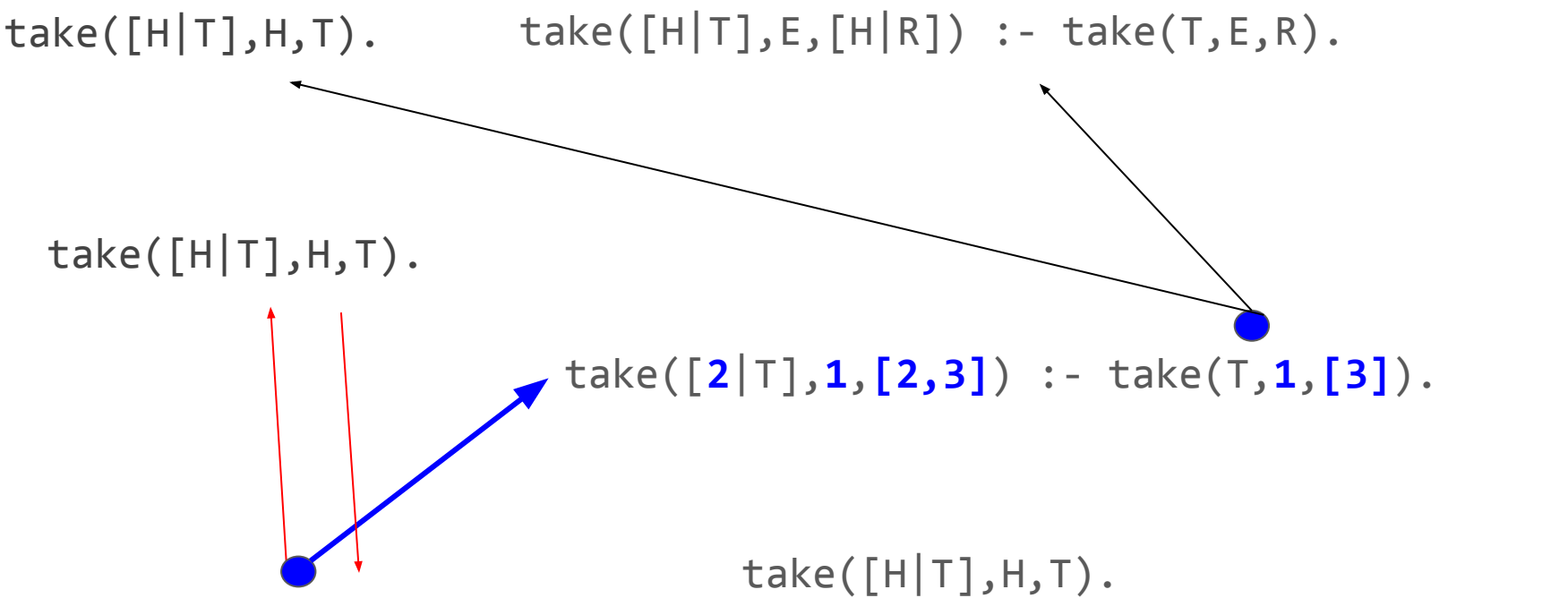
```
take([H|T],H,T).
```

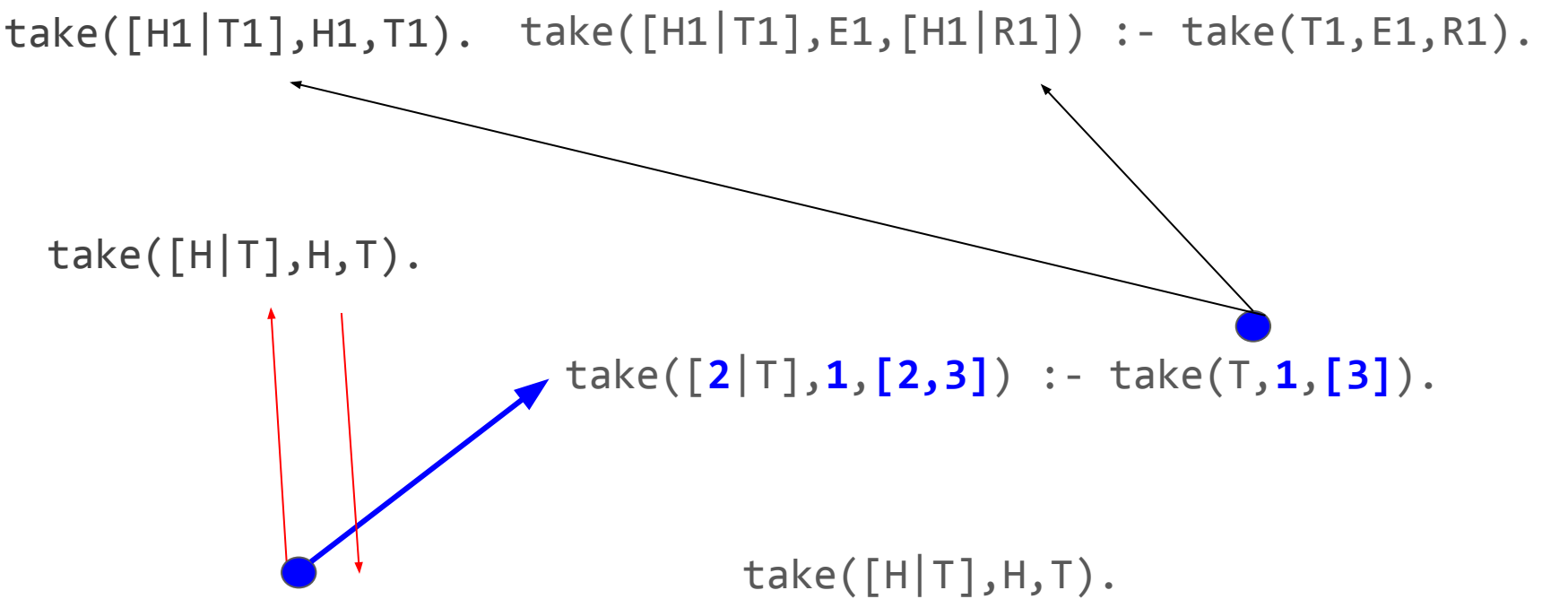


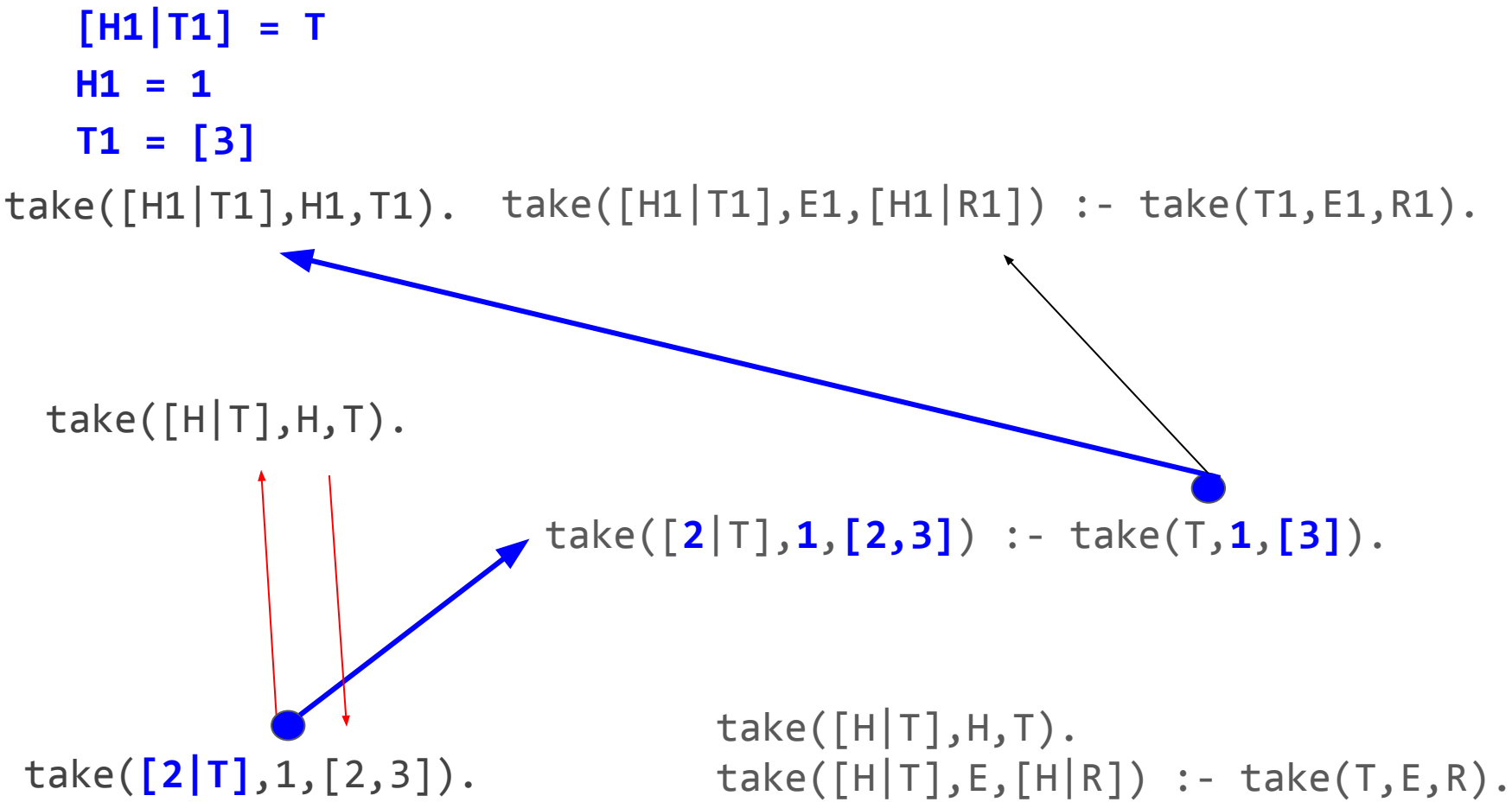
```
take([2|T],1,[2,3]) :- take(T,1,[3]).
```

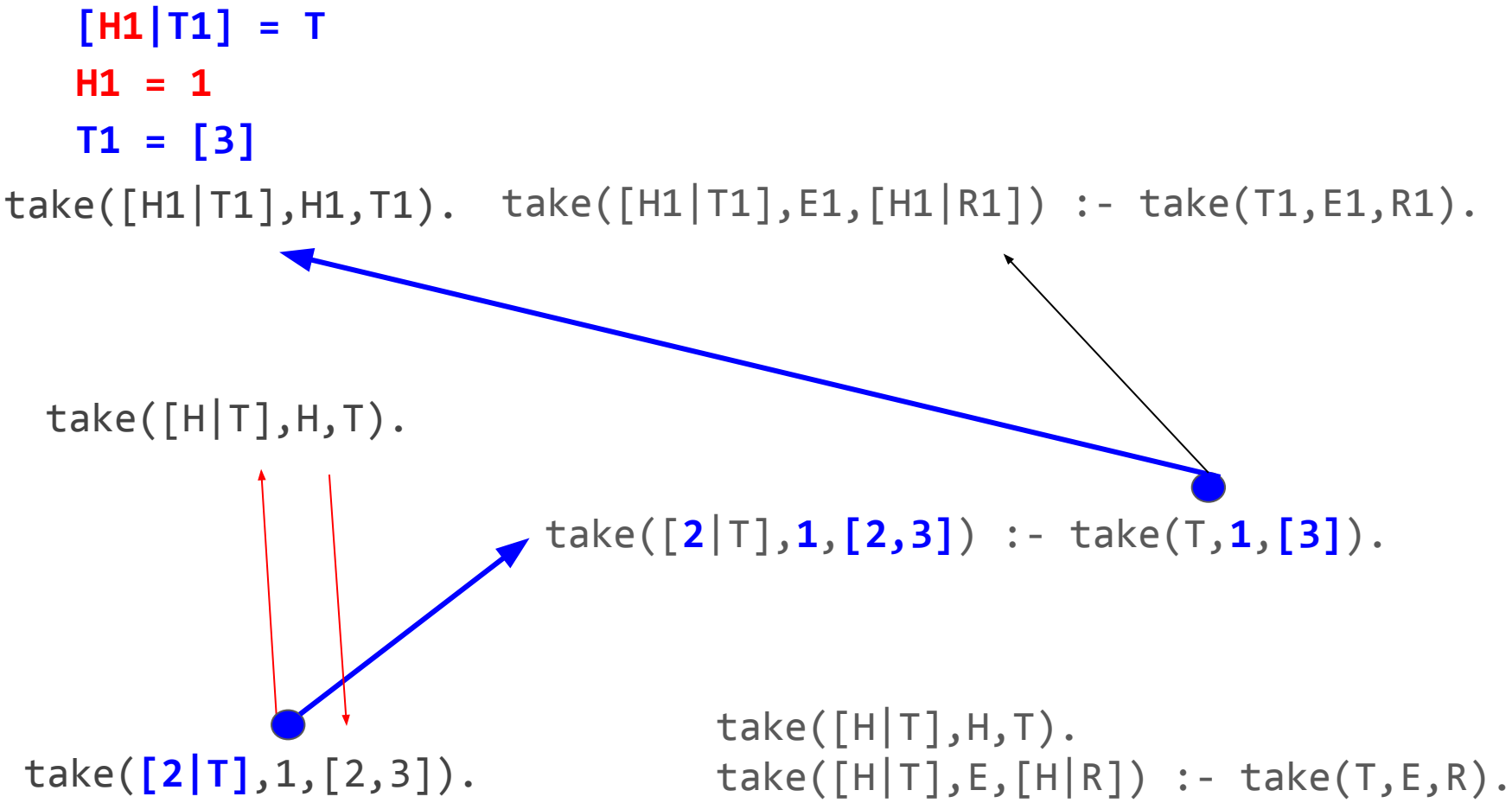
```
take([2|T],1,[2,3]).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```


take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
take([H|T],H,T).
take([2|T],1,[2,3]) :- take(T,1,[3]).
take([2|T],1,[2,3]).
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).


take([H1|T1],H1,T1). take([H1|T1],E1,[H1|R1]) :- take(T1,E1,R1).
take([H|T],H,T).
take([2|T],1,[2,3]) :- take(T,1,[3]).
take([2|T],1,[2,3]).
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).


[H1|T1] = T
H1 = 1
T1 = [3]
take([H1|T1],H1,T1). take([H1|T1],E1,[H1|R1]) :- take(T1,E1,R1).
take([H|T],H,T).
take([2|T],1,[2,3]) :- take(T,1,[3]).
take([2|T],1,[2,3]).
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).


[H1|T1] = T
H1 = 1
T1 = [3]
take([H1|T1],H1,T1). take([H1|T1],E1,[H1|R1]) :- take(T1,E1,R1).
take([H|T],H,T).
take([2|T],1,[2,3]) :- take(T,1,[3]).
take([2|T],1,[2,3]).
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).

[1|T1] = T
T1 = [3]

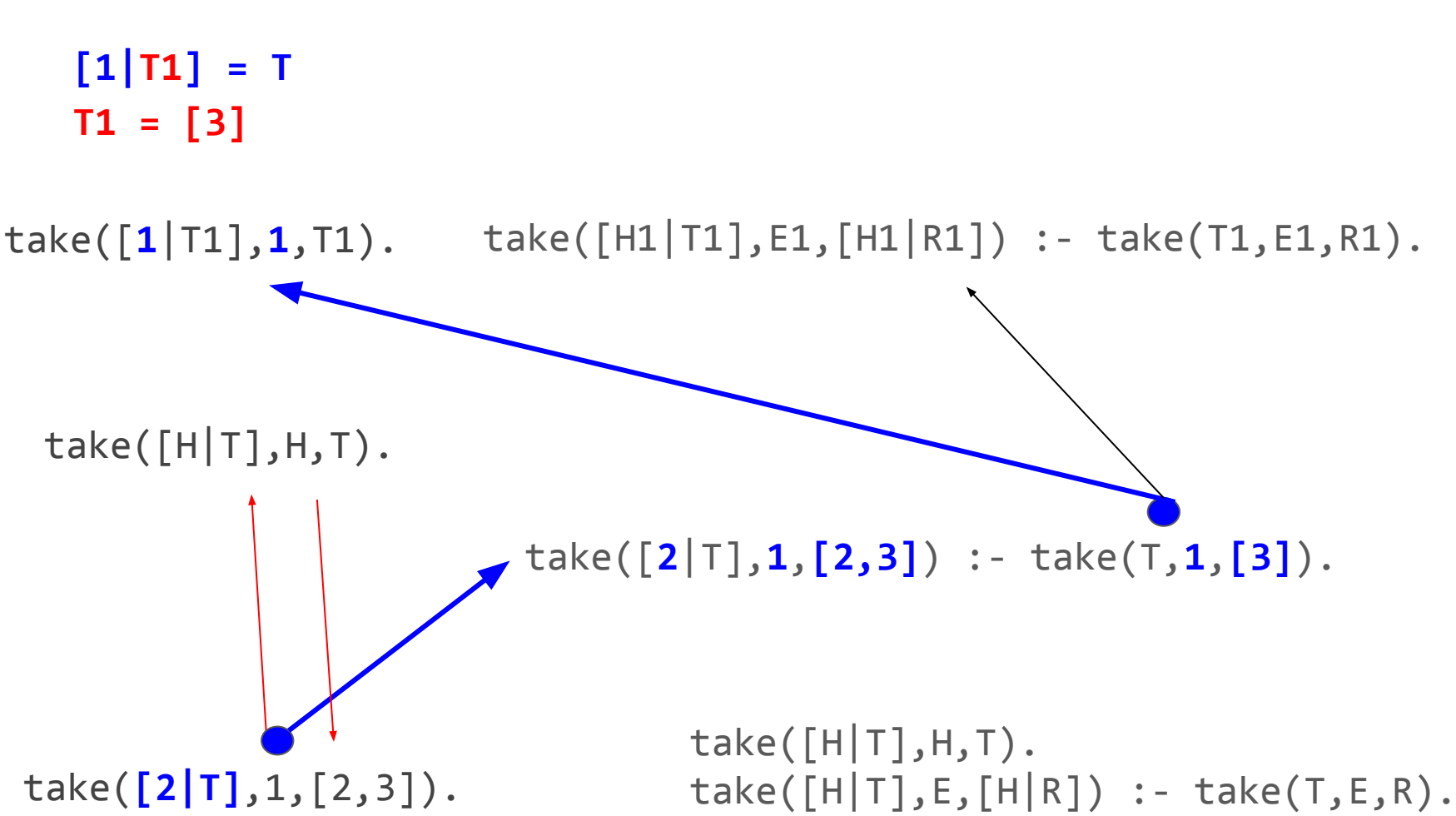
```
take([1|T1],1,T1).    take([H1|T1],E1,[H1|R1]) :- take(T1,E1,R1).
```

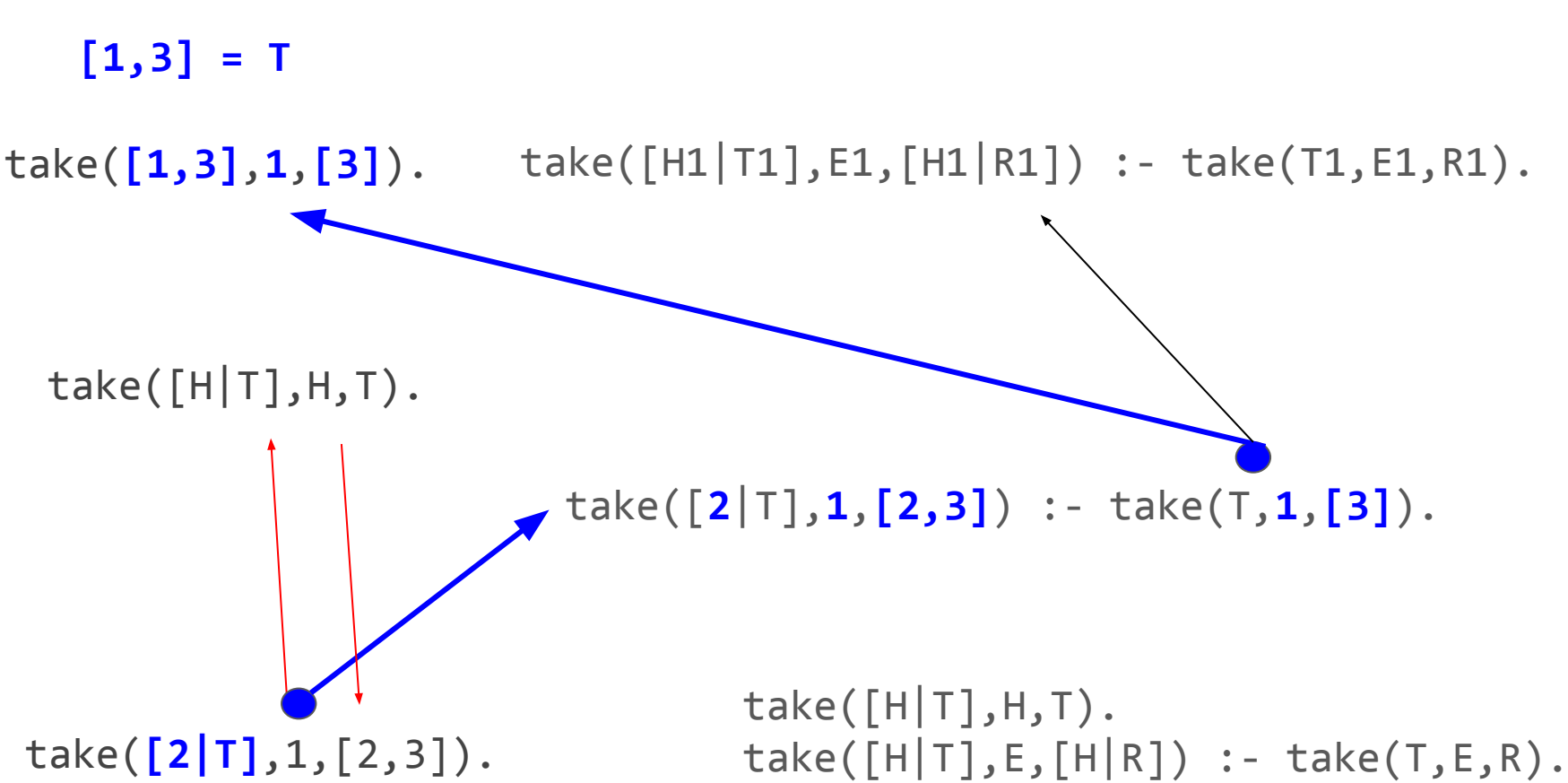
```
take([H|T],H,T).
```

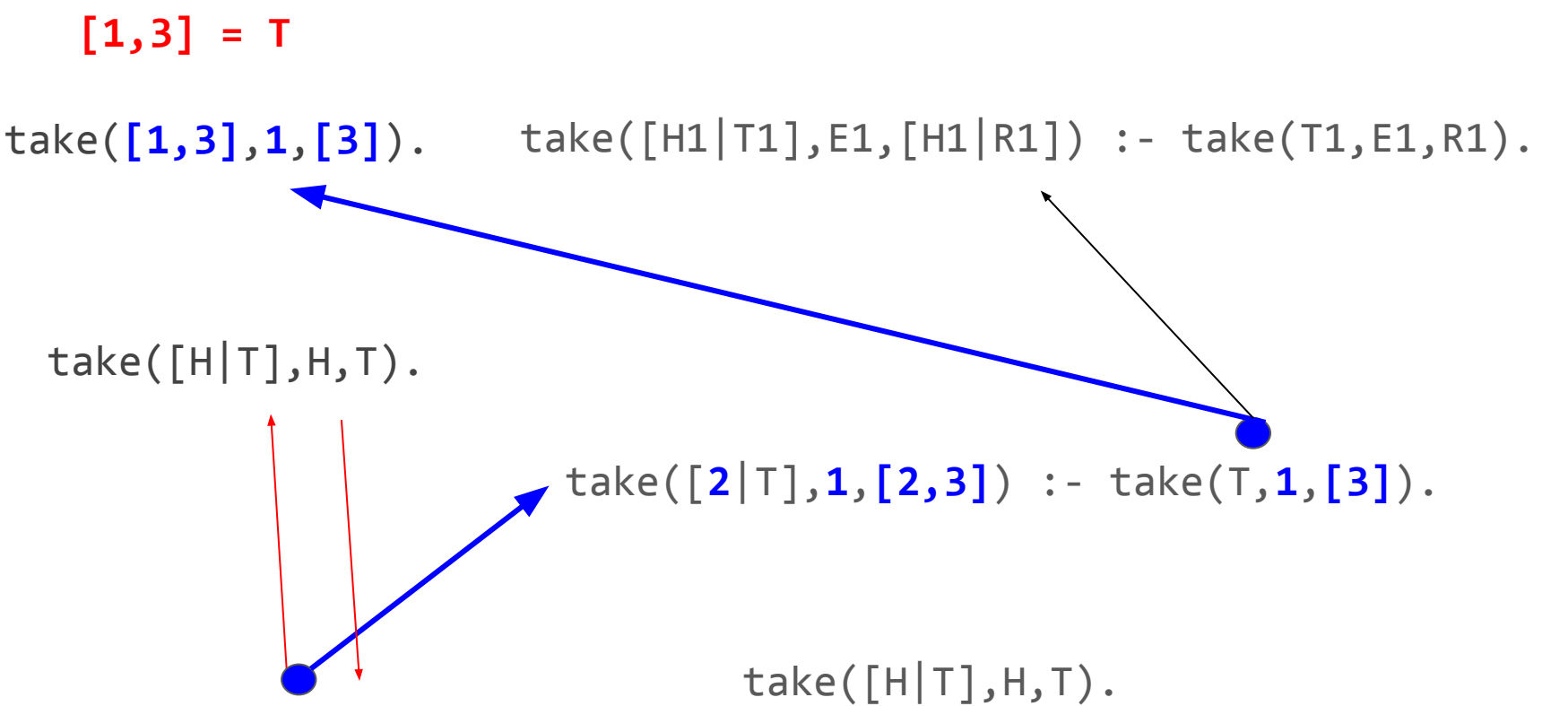
```
take([2|T],1,[2,3]) :- take(T,1,[3]).
```

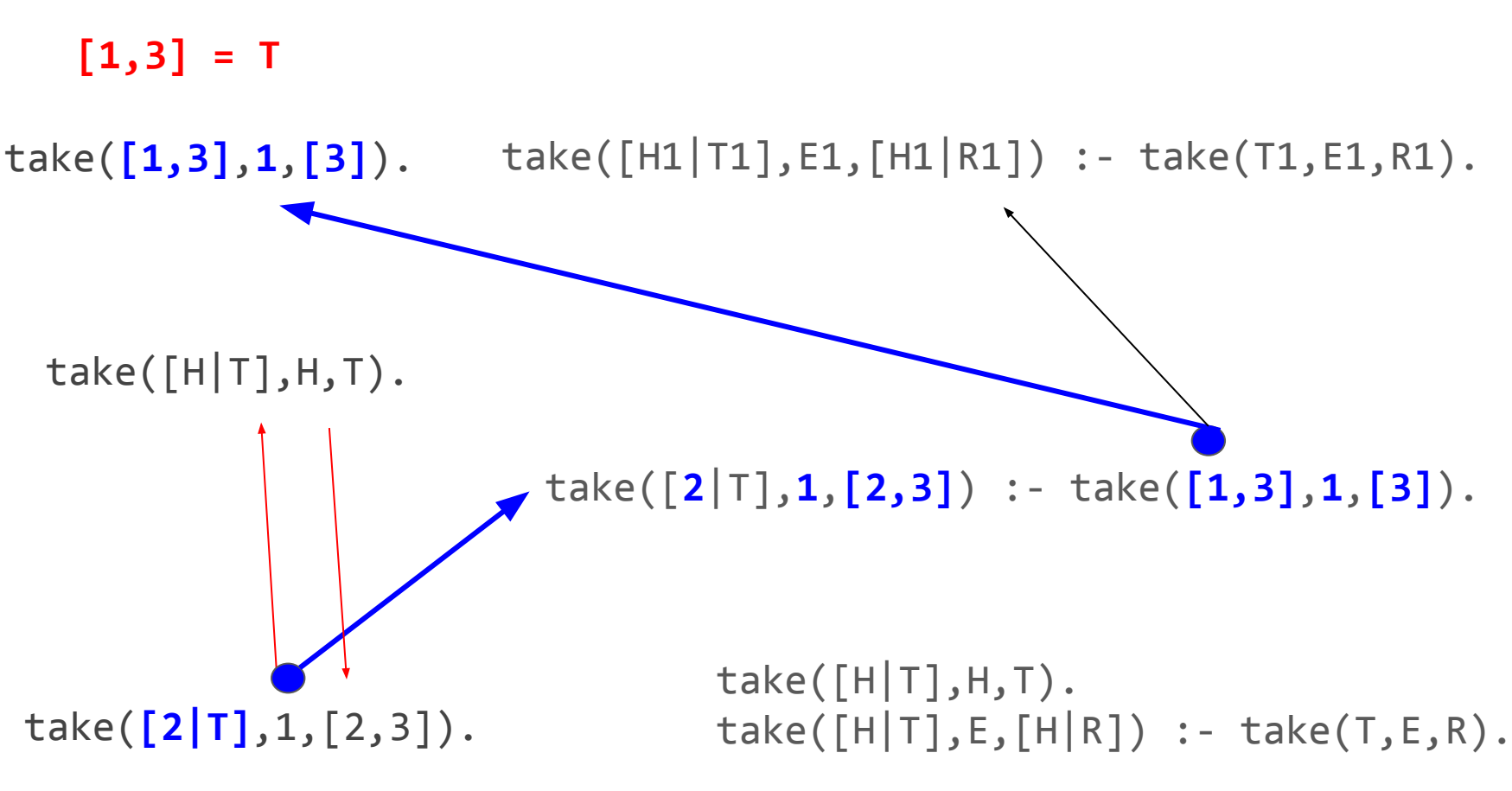
```
take([2|T],1,[2,3]).
```

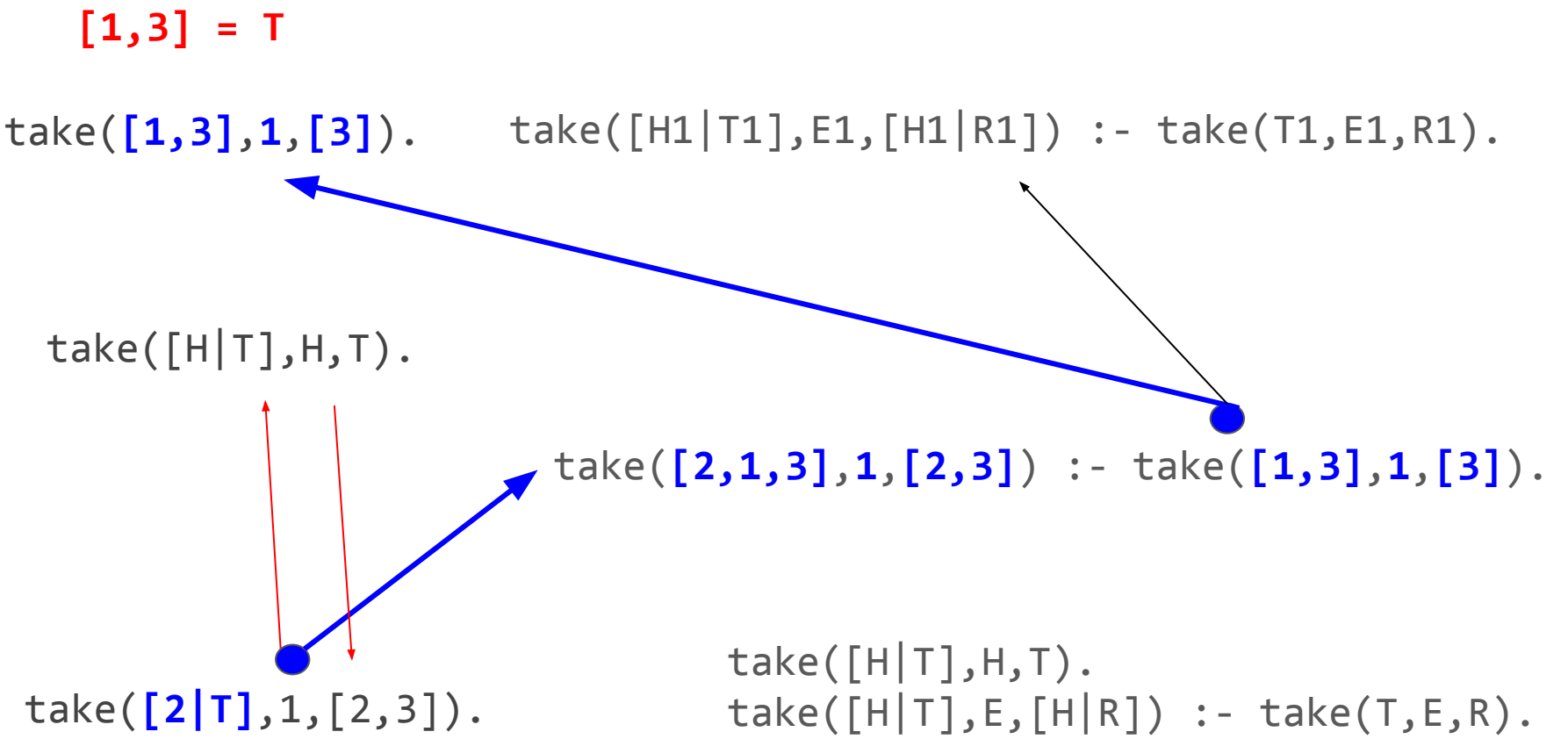
```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

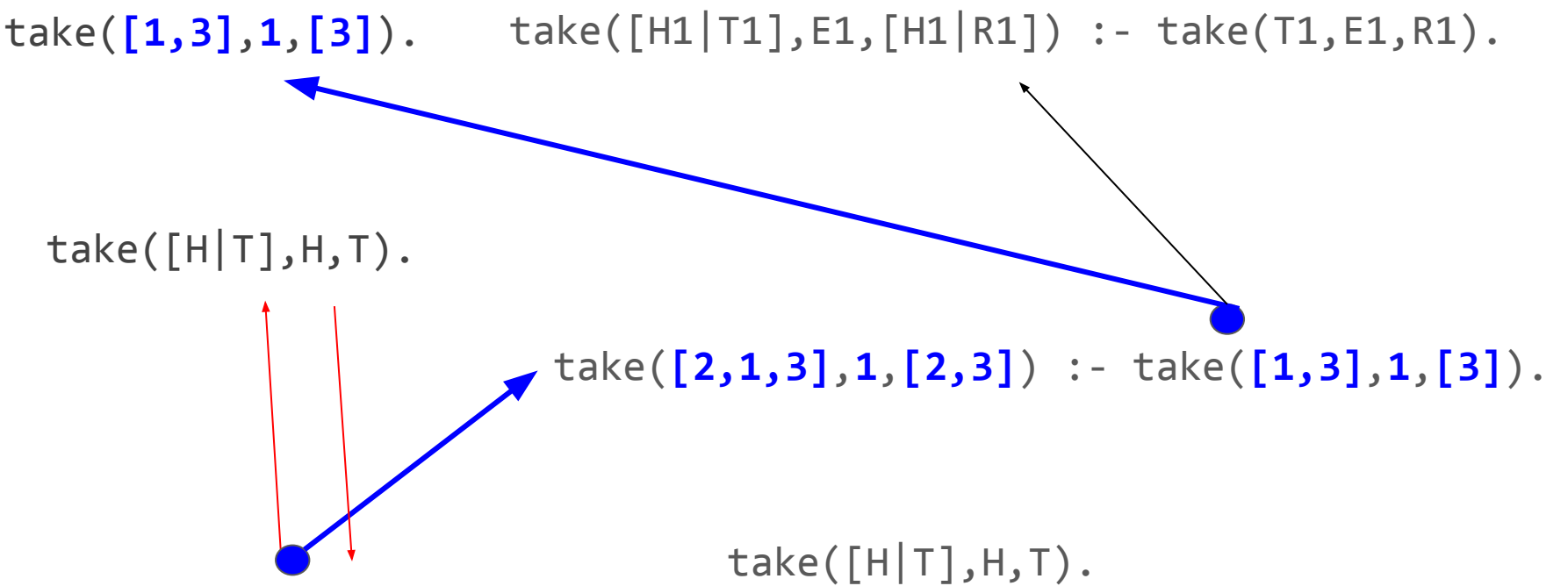

[1|T1] = T
T1 = [3]
take([1|T1],1,T1).
take([H1|T1],E1,[H1|R1]) :- take(T1,E1,R1).
take([H|T],H,T).
take([2|T],1,[2,3]) :- take(T,1,[3]).
take([2|T],1,[2,3]).
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).

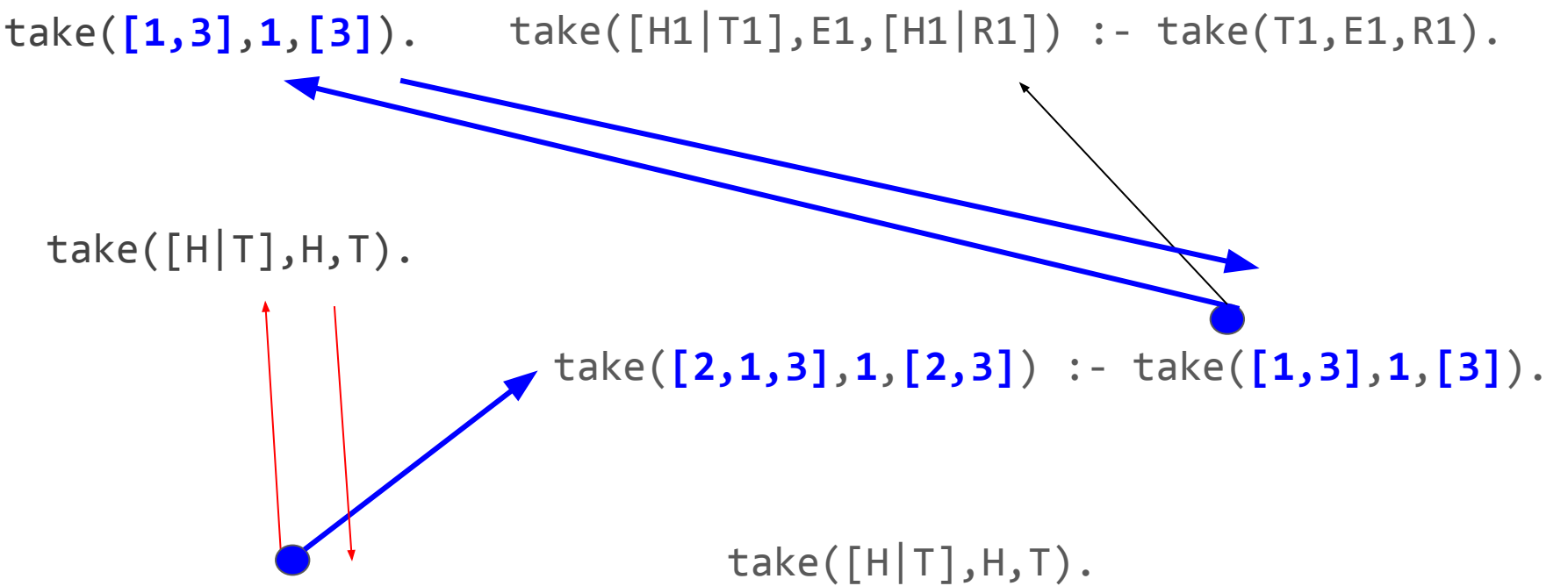

[1,3] = T
take([1,3],1,[3]).
take([H1|T1],E1,[H1|R1]) :- take(T1,E1,R1).
take([H|T],H,T).
take([2|T],1,[2,3]) :- take(T,1,[3]).
take([2|T],1,[2,3]).
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).


[1,3] = T
take([1,3],1,[3]).
take([H1|T1],E1,[H1|R1]) :- take(T1,E1,R1).
take([H|T],H,T).
take([2|T],1,[2,3]) :- take(T,1,[3]).
take([2|T],1,[2,3]).
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).


[1,3] = T
take([1,3],1,[3]).
take([H1|T1],E1,[H1|R1]) :- take(T1,E1,R1).
take([H|T],H,T).
take([2|T],1,[2,3]) :- take([1,3],1,[3]).
take([2|T],1,[2,3]).
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).


[1,3] = T
take([1,3],1,[3]).
take([H1|T1],E1,[H1|R1]) :- take(T1,E1,R1).
take([H|T],H,T).
take([2,1,3],1,[2,3]) :- take([1,3],1,[3]).
take([2|T],1,[2,3]).
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).

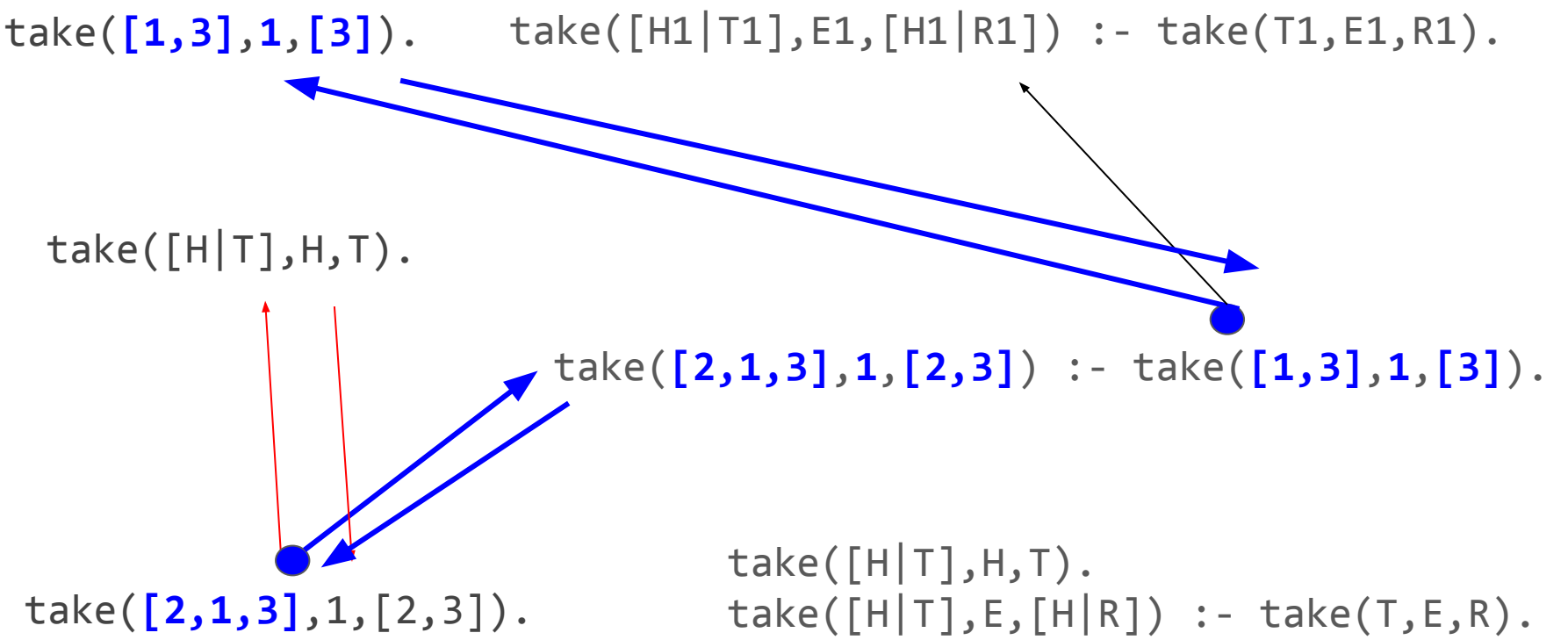

take([1,3],1,[3]).
take([H1|T1],E1,[H1|R1]) :- take(T1,E1,R1).
take([H|T],H,T).
take([2,1,3],1,[2,3]) :- take([1,3],1,[3]).
take([2,1,3],1,[2,3]).
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).


take([1,3],1,[3]).
take([H1|T1],E1,[H1|R1]) :- take(T1,E1,R1).
take([H|T],H,T).
take([2,1,3],1,[2,3]) :- take([1,3],1,[3]).
take([2,1,3],1,[2,3]).
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).

http://etc.ch/8WDC

DirectPoll

Q: finished taking notes?

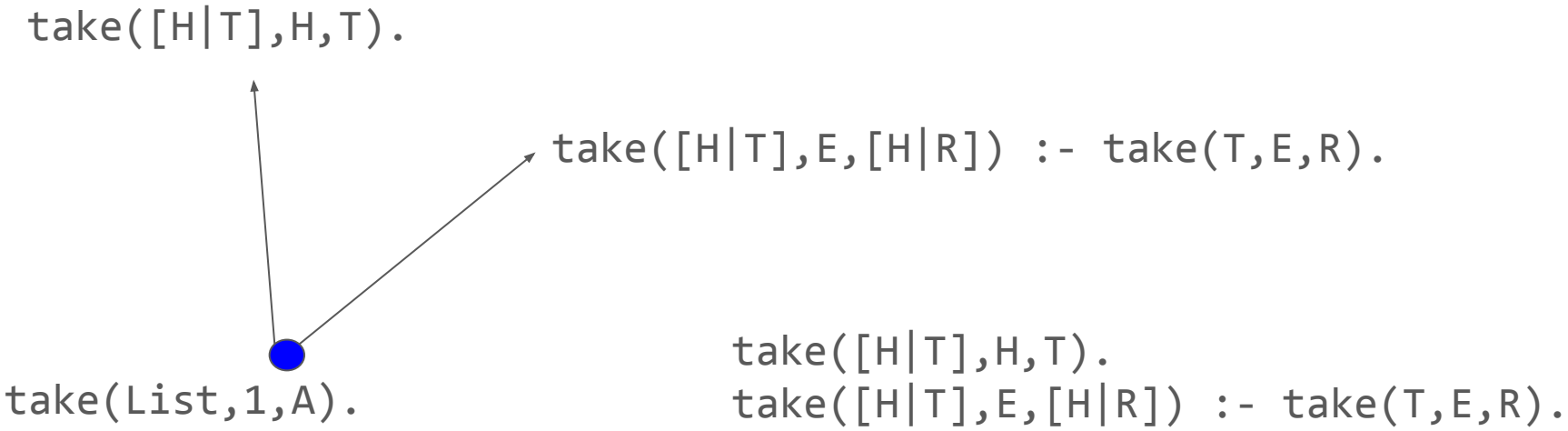


What happens if we try take(-List,+E,-R)?

```
take(List,1,A).
```

```
take(List,1,A).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

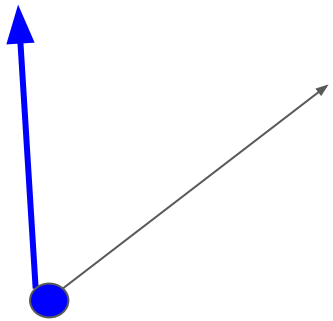

take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
take(List,1,A).
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).

[H|T] = List

H = 1

T = A

```
take([H|T],H,T).
```



```
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take(List,1,A).
```

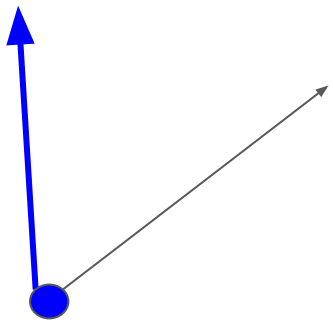
```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

[H|T] = List

H = 1

T = A

```
take([H|T],H,T).
```



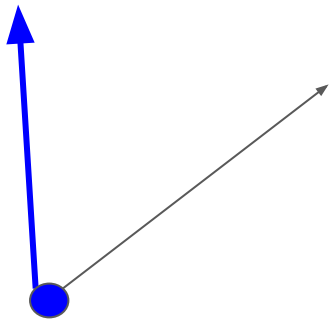
```
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take(List,1,A).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

[1|T] = List
T = A

```
take([1|T],1,T).
```



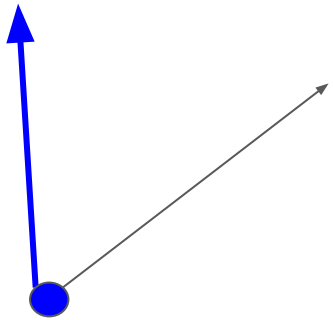
```
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take(List,1,A).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

[1|T] = List
T = A

```
take([1|T],1,T).
```



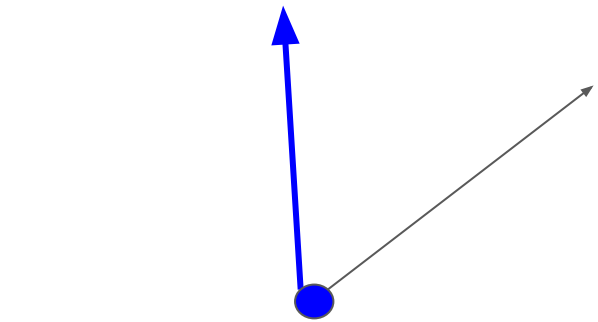
```
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take(List,1,A).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

[1|A] = List

```
take([1|A],1,A).
```



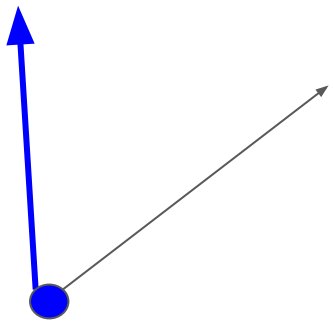
```
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take(List,1,A).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

[1|A] = List

```
take([1|A],1,A).
```

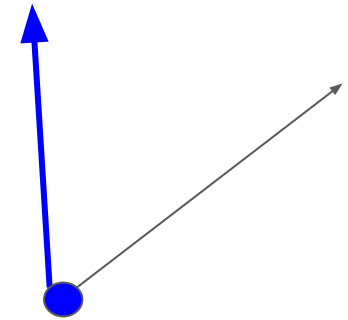


```
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take(List,1,A).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

take([1|A],1,A).

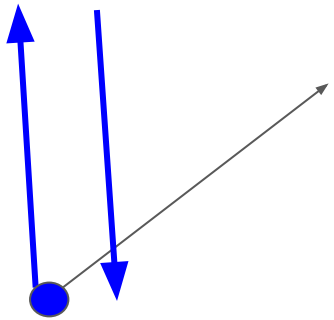


take([H|T],E,[H|R]) :- take(T,E,R).

take([1|A],1,A).

take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).

```
take([1|A],1,A).
```

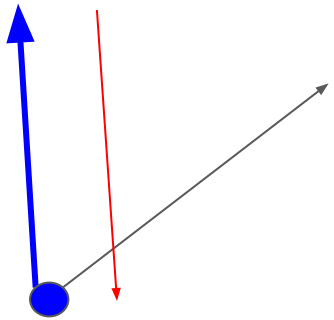


```
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take([1|A],1,A).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take([1|A],1,A).
```

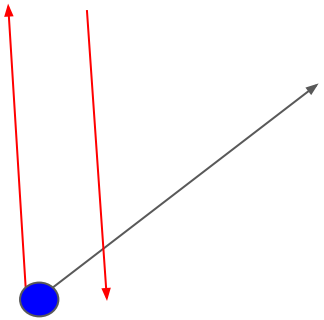


```
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take([1|A],1,A).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take([H|T],H,T).
```



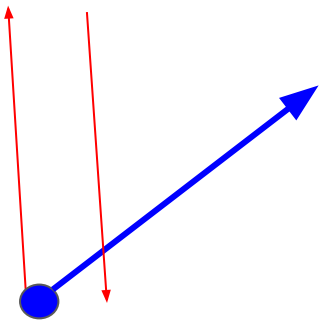
```
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take(List,1,A).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

[H|T] = List
E = 1
[H|R] = A

```
take([H|T],H,T).
```



```
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take(List,1,A).
```

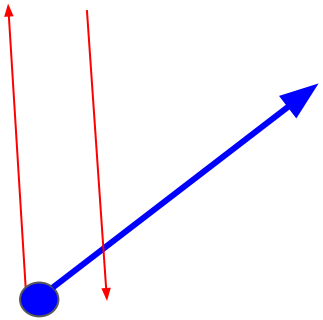
```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

[H|T] = List

E = 1

[H|R] = A

```
take([H|T],H,T).
```



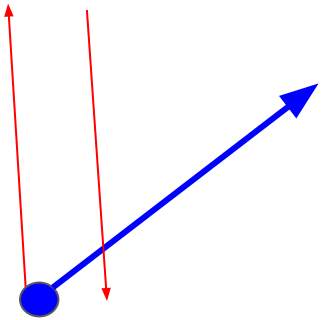
```
take([H|T],E,[H|R]) :- take(T,E,R).
```

```
take(List,1,A).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

[H|T] = List
[H|R] = A

```
take([H|T],H,T).
```



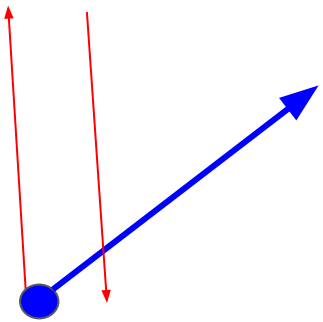
```
take([H|T],1,[H|R]) :- take(T,1,R).
```

```
take(List,1,A).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

[H|T] = List
[H|R] = A

```
take([H|T],H,T).
```



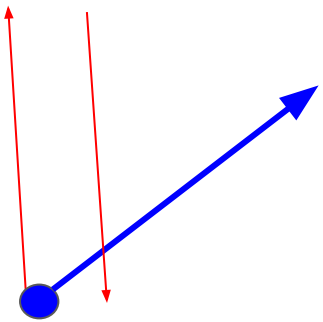
```
take([H|T],1,[H|R]) :- take(T,1,R).
```

```
take(List,1,A).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

[H|T] = List

```
take([H|T],H,T).
```



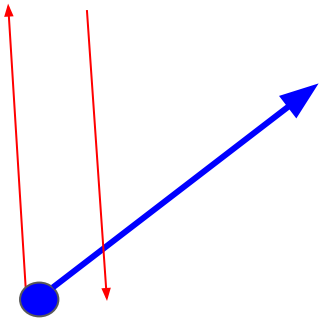
```
take([H|T],1,[H|R]) :- take(T,1,R).
```

```
take(List,1,[H|R]).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

[H|T] = List

```
take([H|T],H,T).
```

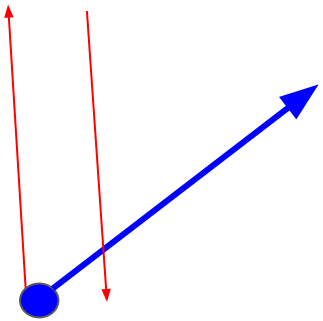


```
take([H|T],1,[H|R]) :- take(T,1,R).
```

```
take(List,1,[H|R]).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

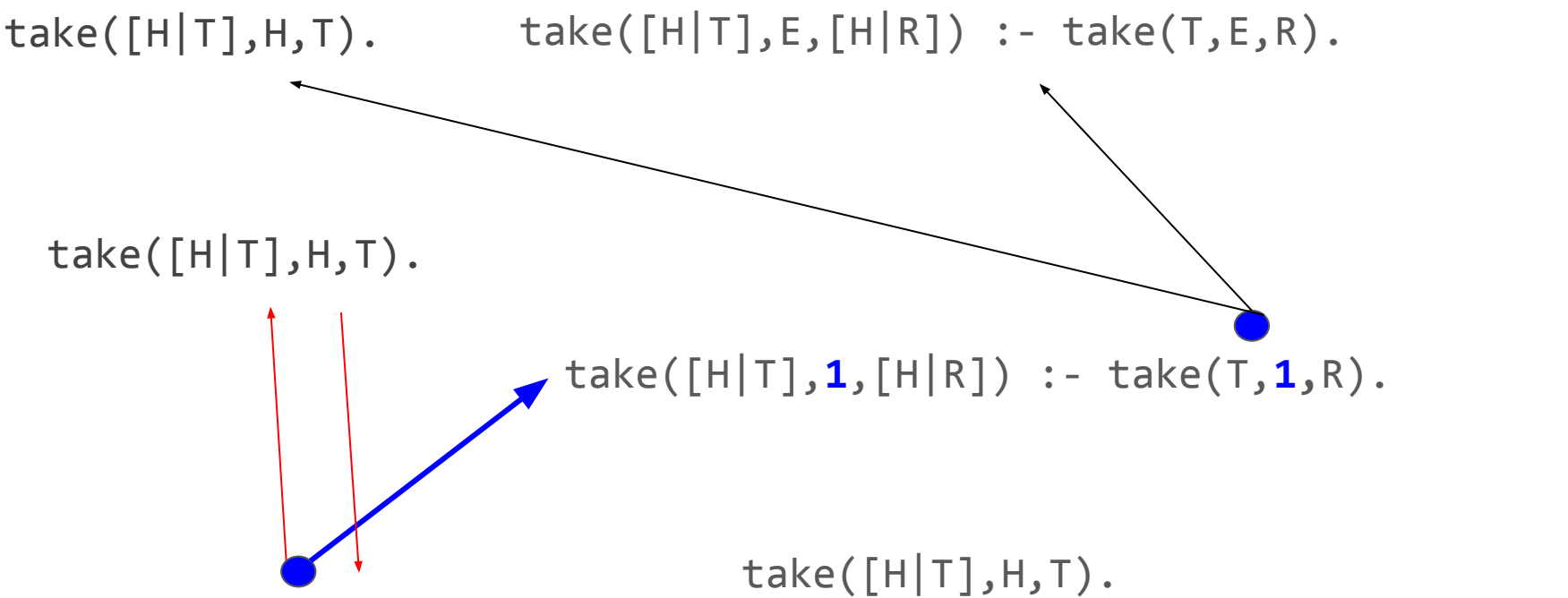
```
take([H|T],H,T).
```

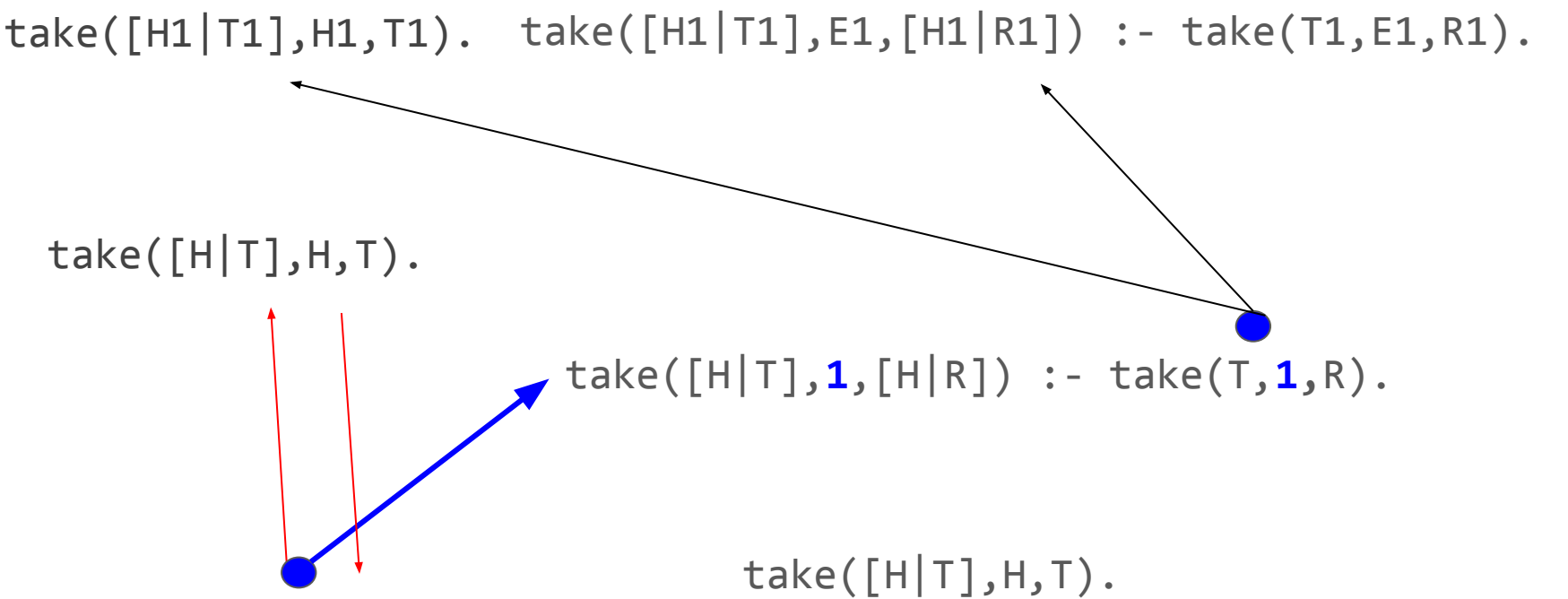


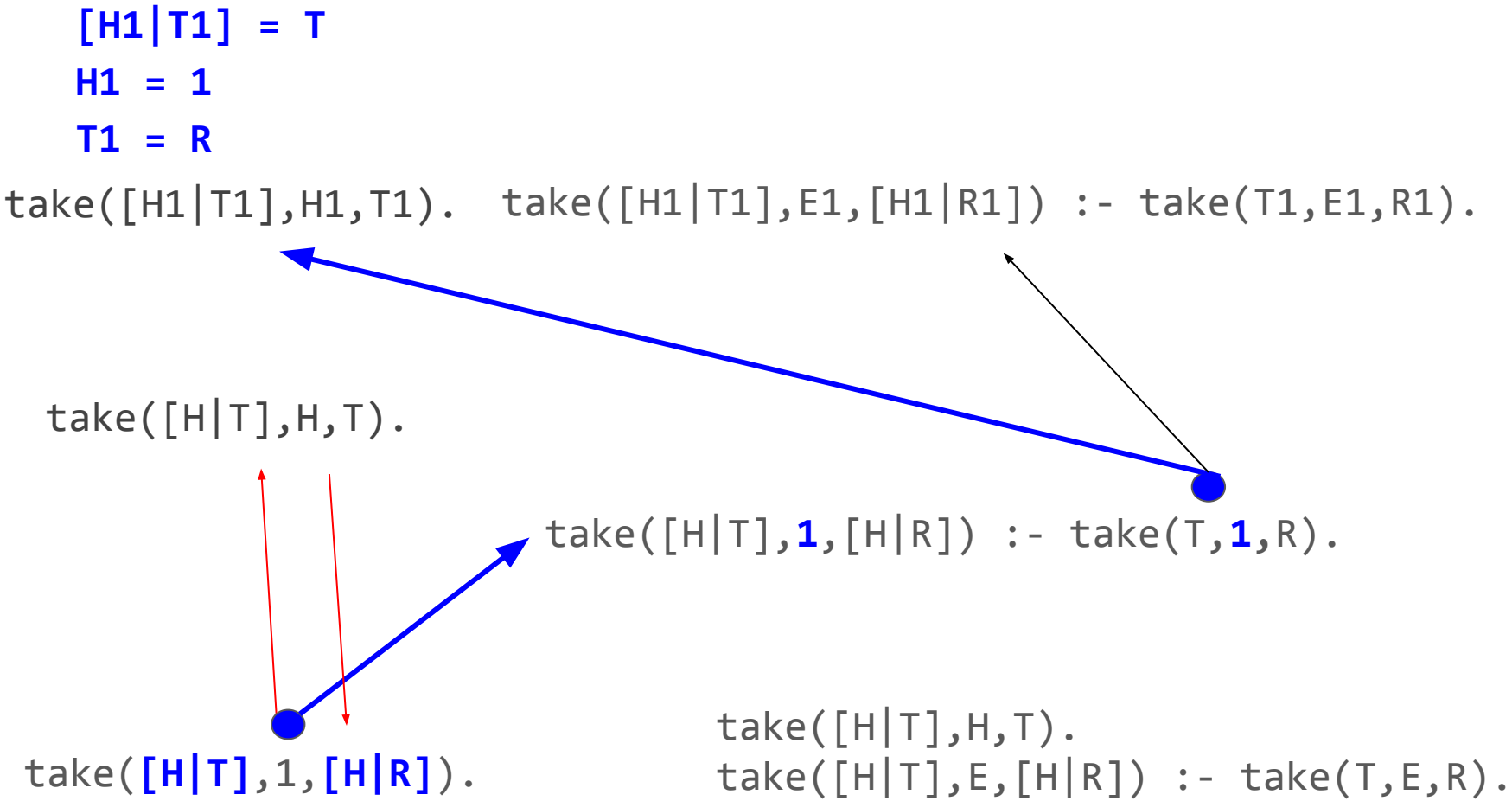
```
take([H|T],1,[H|R]) :- take(T,1,R).
```

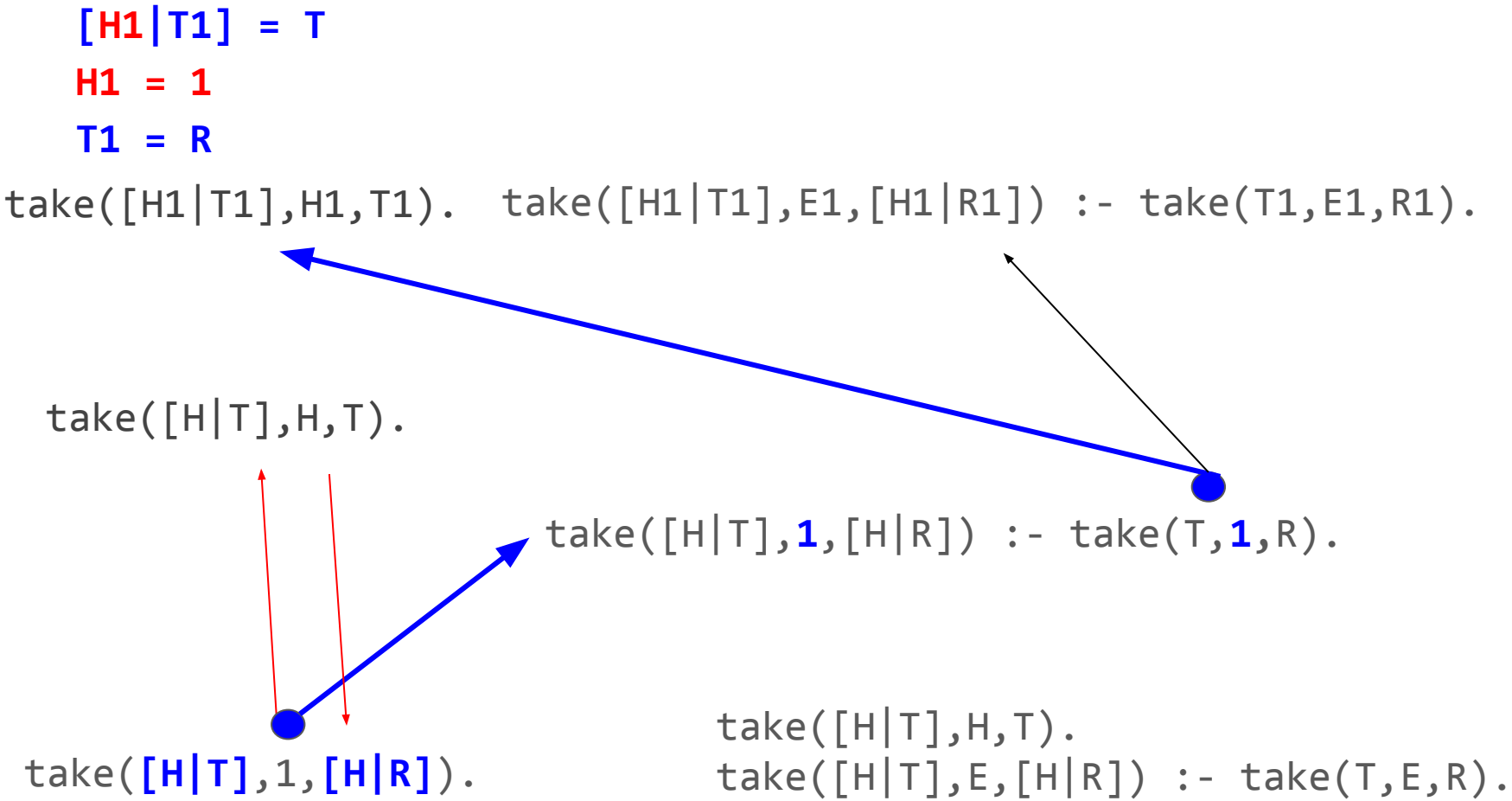
```
take([H|T],1,[H|R]).
```

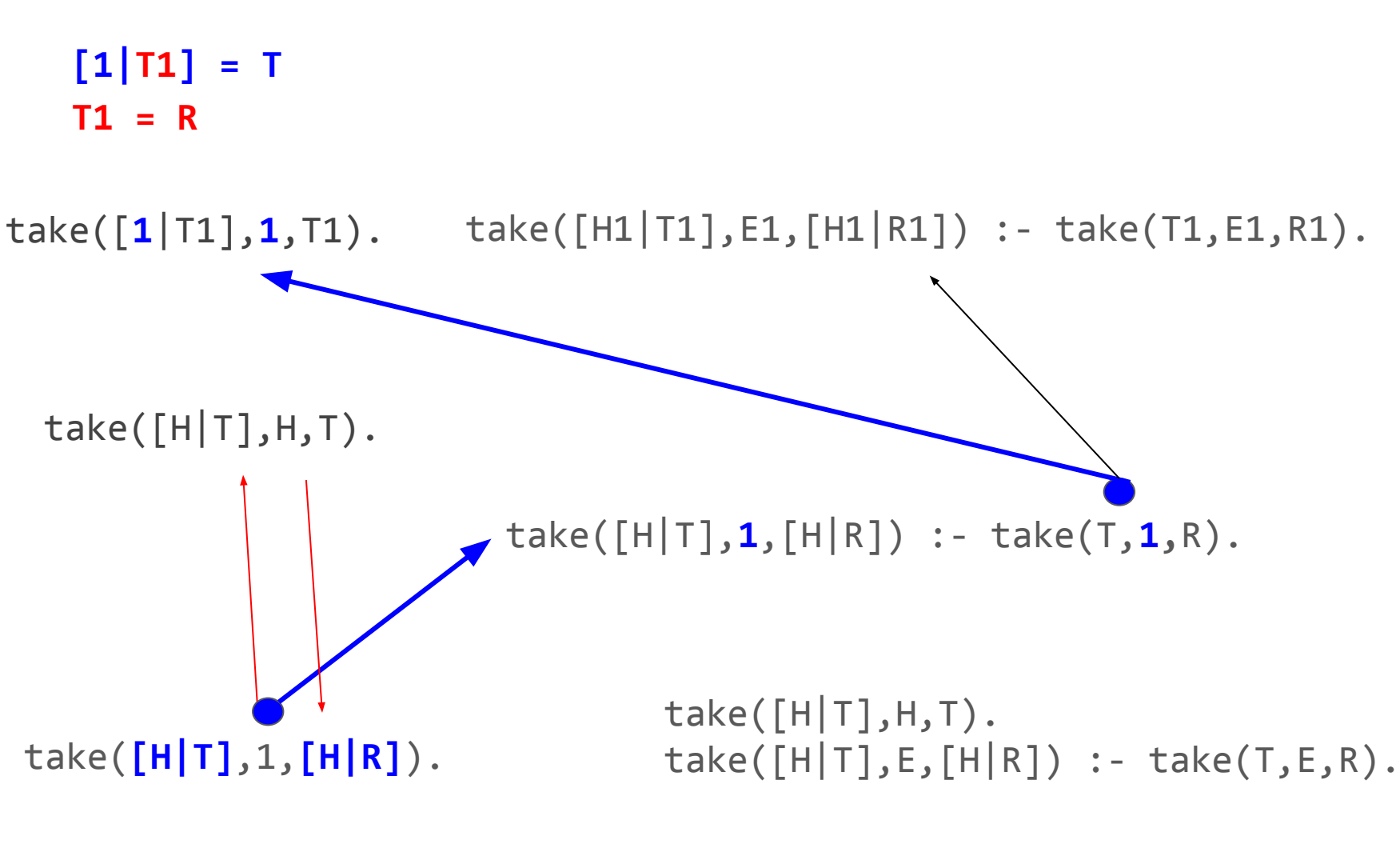
```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```


take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
take([H|T],H,T).
take([H|T],1,[H|R]) :- take(T,1,R).
take([H|T],1,[H|R]).
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).


take([H1|T1],H1,T1). take([H1|T1],E1,[H1|R1]) :- take(T1,E1,R1).
take([H|T],H,T).
take([H|T],1,[H|R]) :- take(T,1,R).
take([H|T],1,[H|R]).
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).


[H1|T1] = T
H1 = 1
T1 = R
take([H1|T1],H1,T1). take([H1|T1],E1,[H1|R1]) :- take(T1,E1,R1).
take([H|T],H,T).
take([H|T],1,[H|R]) :- take(T,1,R).
take([H|T],1,[H|R]).
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).


[H1|T1] = T
H1 = 1
T1 = R
take([H1|T1],H1,T1). take([H1|T1],E1,[H1|R1]) :- take(T1,E1,R1).
take([H|T],H,T).
take([H|T],1,[H|R]) :- take(T,1,R).
take([H|T],1,[H|R]).
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).


[1|T1] = T
T1 = R
take([1|T1],1,T1).
take([H1|T1],E1,[H1|R1]) :- take(T1,E1,R1).
take([H|T],H,T).
take([H|T],1,[H|R]) :- take(T,1,R).
take([H|T],1,[H|R]).
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).

[1|R] = T

```
take([1|R],1,R).
```

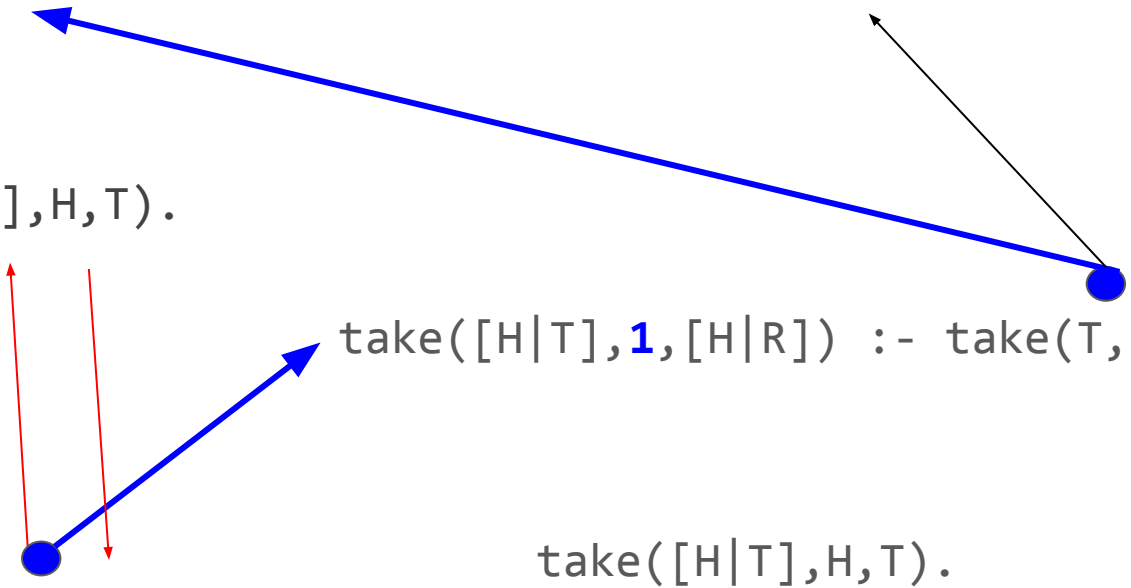
```
take([H1|T1],E1,[H1|R1]) :- take(T1,E1,R1).
```

```
take([H|T],H,T).
```

```
take([H|T],1,[H|R]) :- take(T,1,R).
```

```
take([H|T],1,[H|R]).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```



[1|R] = T

```
take([1|R],1,R).
```

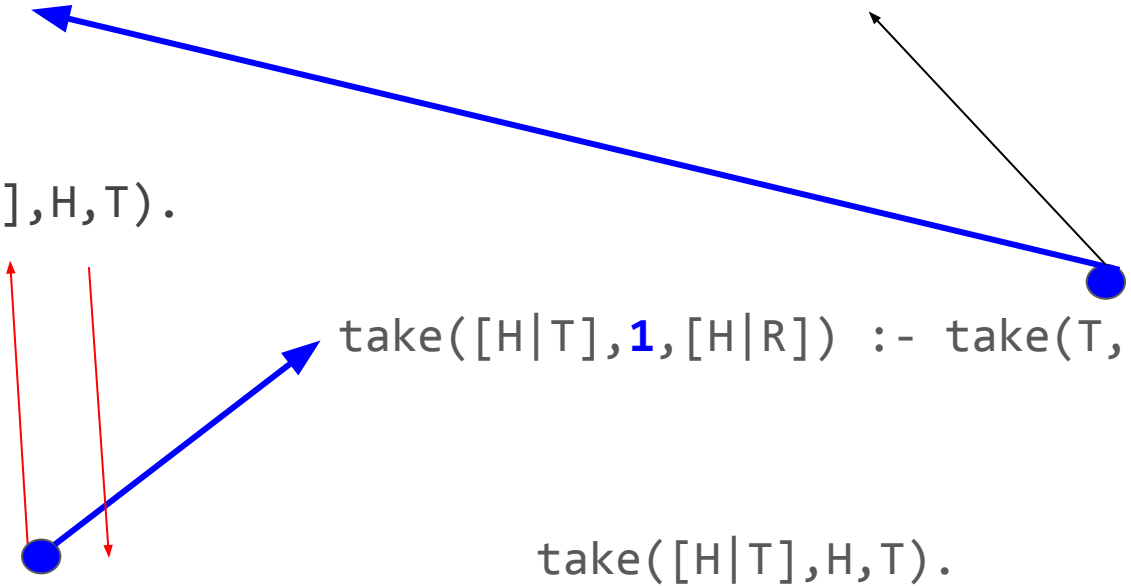
```
take([H1|T1],E1,[H1|R1]) :- take(T1,E1,R1).
```

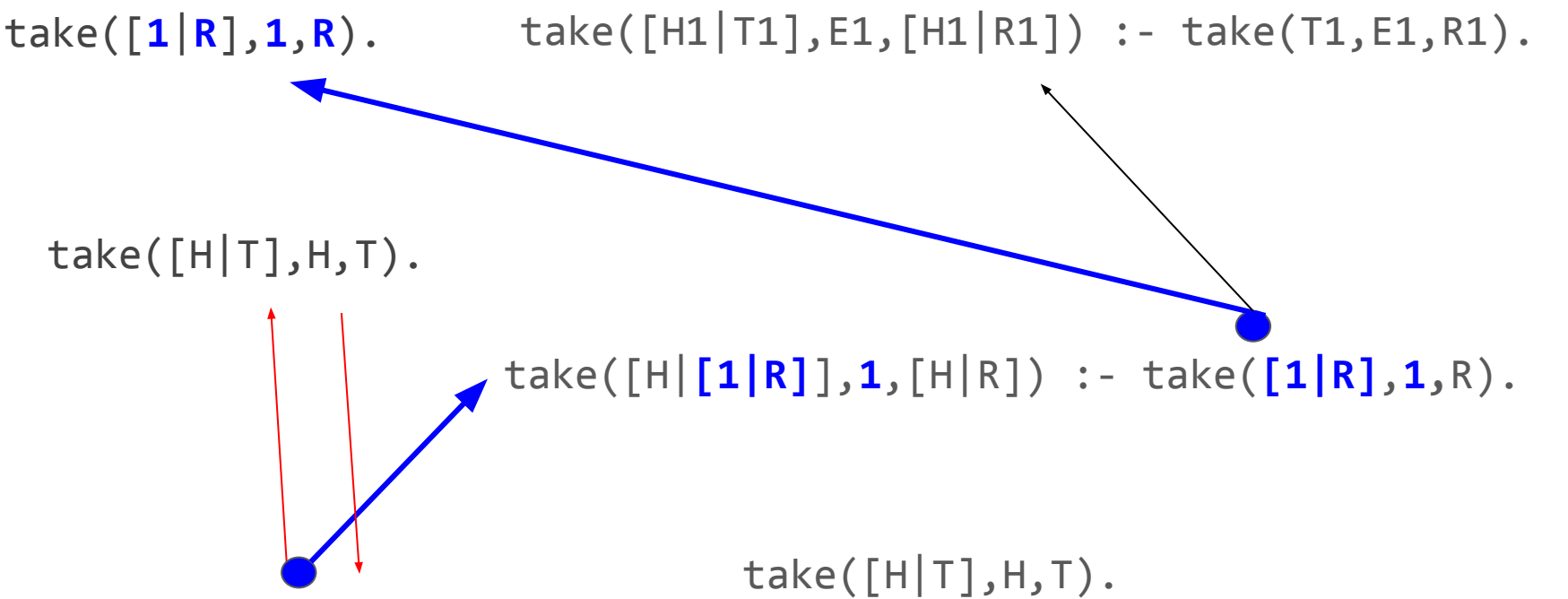
```
take([H|T],H,T).
```

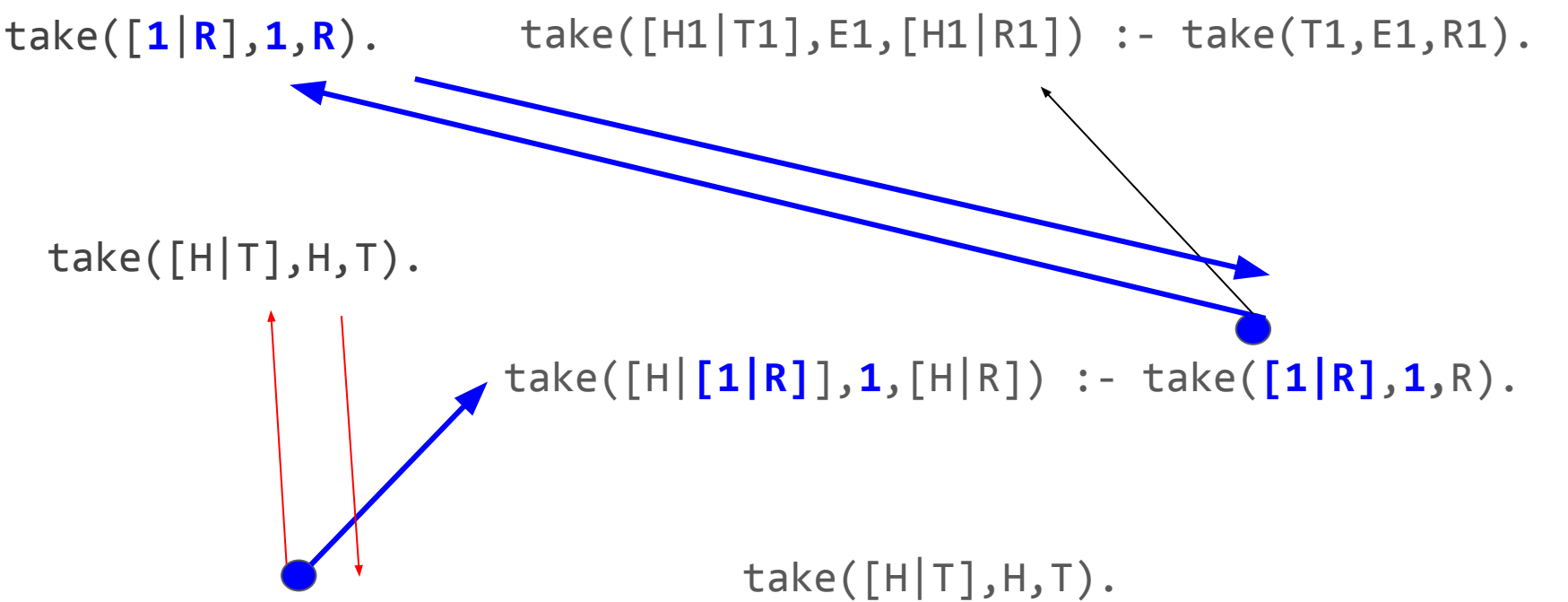
```
take([H|T],1,[H|R]) :- take(T,1,R).
```

```
take([H|T],1,[H|R]).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```




take([1|R],1,R).
take([H1|T1],E1,[H1|R1]) :- take(T1,E1,R1).
take([H|T],H,T).
take([H|[1|R]],1,[H|R]) :- take([1|R],1,R).
take([H|[1|R]],1,[H|R]).
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).


take([1|R],1,R).
take([H1|T1],E1,[H1|R1]) :- take(T1,E1,R1).
take([H|T],H,T).
take([H|[1|R]],1,[H|R]) :- take([1|R],1,R).
take([H|[1|R]],1,[H|R]).
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).

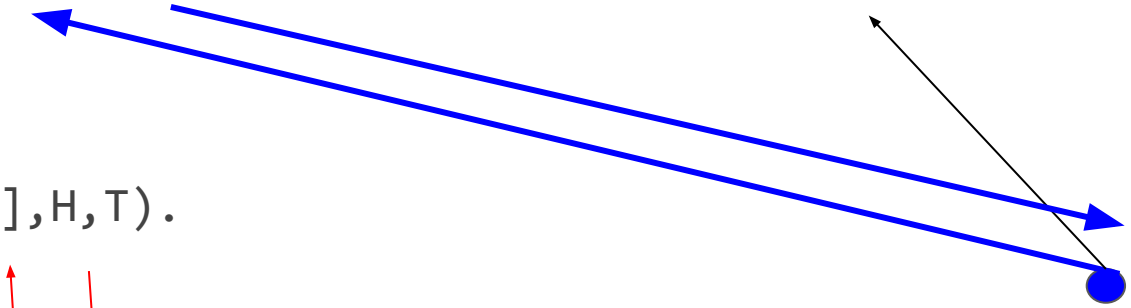
http://etc.ch/8WDC

DirectPoll

Q: finished taking notes?

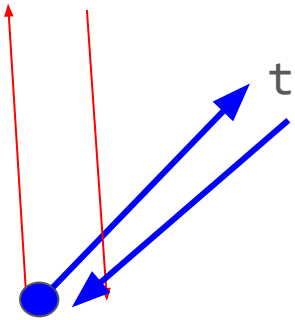
```
take([1|R],1,R).
```

```
take([H1|T1],E1,[H1|R1]) :- take(T1,E1,R1).
```



```
take([H|T],H,T).
```

```
take([H|[1|R]],1,[H|R]) :- take([1|R],1,R).
```



```
take([H|[1|R]],1,[H|R]).
```

```
take([H|T],H,T).
take([H|T],E,[H|R]) :- take(T,E,R).
```

Recap: first result

```
?- take(List,[1],A).

List = [1|A]
```

Recap: second result

```
?- take(List,[1],A).

List = [1|A] ;

List = [H,1|R], A = [H|R]          (prolog writes this as [H|[1|R]])
```

Recap: third result?

```
?- take(List,[1],A).

List = [1|A] ;

List = [H,1|R], A = [H|R] ;      (prolog writes this as [H|[1|R]])

List = ...
```

http://etc.ch/8WDC

DirectPoll

Every time you backtrack it finds a longer list...

```
?- take(List,[1],A).

List = [1|A] ;

List = [H,1|R], A = [H|R] ;

List = [H1,H2,1|R], A = [H1,H2|R] ;

List = [H1,H2,H3,1|R], A = [H1,H2,H3|R] ;

List = [H1,H2,H3,H4,1|R], A = [H1,H2,H3,H4|R] ;
```

Q: I often write logically-correct code which doesn't terminate. What heuristics can I apply to see if this will happen without running the code?

A: Also think about termination in terms of the backtracking search...

Implement perm(+A,-B).

perm(+A,-B) succeeds if the list B is a permutation of the list A.

http://etc.ch/8WDC

Implement perm(+A,-B).

```
take([H|T],H,T).
take([H|T],R,[H|S]) :- take(T,R,S).

perm([],[]).   % permutation of the empty list is an empty list


...
```

Implement perm(+A,-B).

```
take([H|T],H,T).
take([H|T],R,[H|S]) :- take(T,R,S).

perm([],[]).   % permutation of the empty list is an empty list

% permutation of list L1 is [E1|L2] if L2 is a permutation of
% what's left after you take E1 from L1.
perm(L1,[E1|L2]) :- take(L1,E1,R1), perm(R1,L2).
```

perm(+A,-B) is 'classic' Prolog

```
Learn it :-)
```

Q: I often write logically-correct code which doesn't terminate. What heuristics can I apply to see if this will happen without running the code?

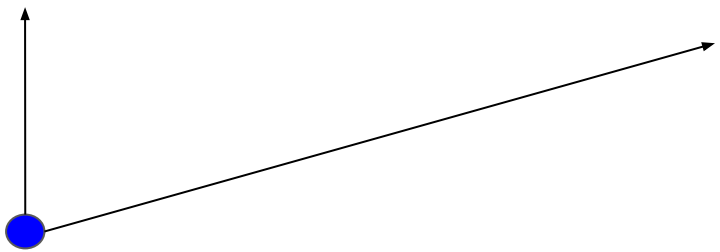
What happens if we use perm 'backwards'

```
perm(L,[1]).
```

```
perm(L,[1]).
```

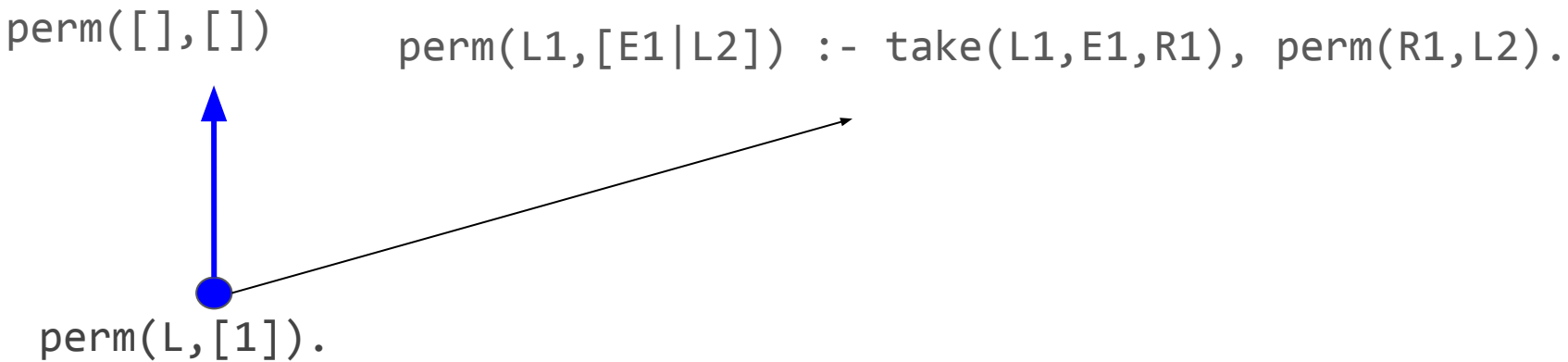
```
perm([],[])
.

perm(L1,[E1|L2]) :- take(L1,E1,R1), perm(R1,L2).
```



```
perm(L,[1]).
```

[] = L
[] = [1]

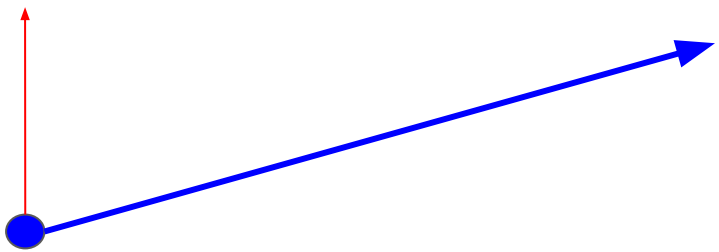


L1 = L

[E1|L2] = [1|[]]

```
perm([],[])

perm(L1,[E1|L2]) :- take(L1,E1,R1), perm(R1,L2).
```



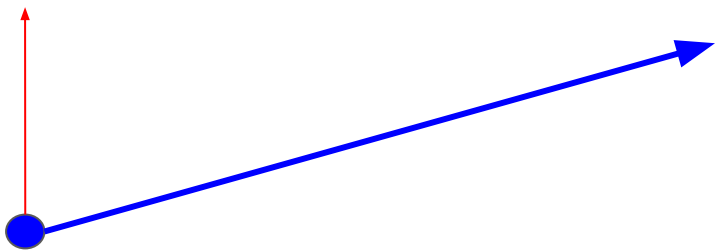
```
perm(L,[1]).
```

L1 = L

[E1|L2] = [1|[]]

```
perm([],[])

perm(L1,[E1|L2]) :- take(L1,E1,R1), perm(R1,L2).
```

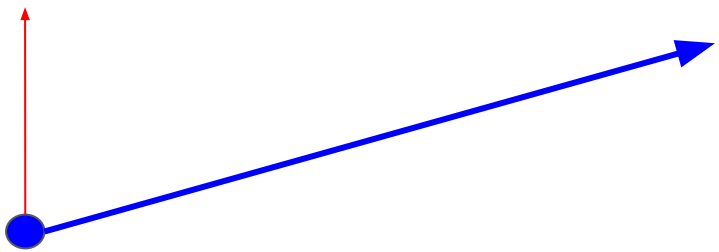


```
perm(L,[1]).
```

[E1|L2] = [1|[]]

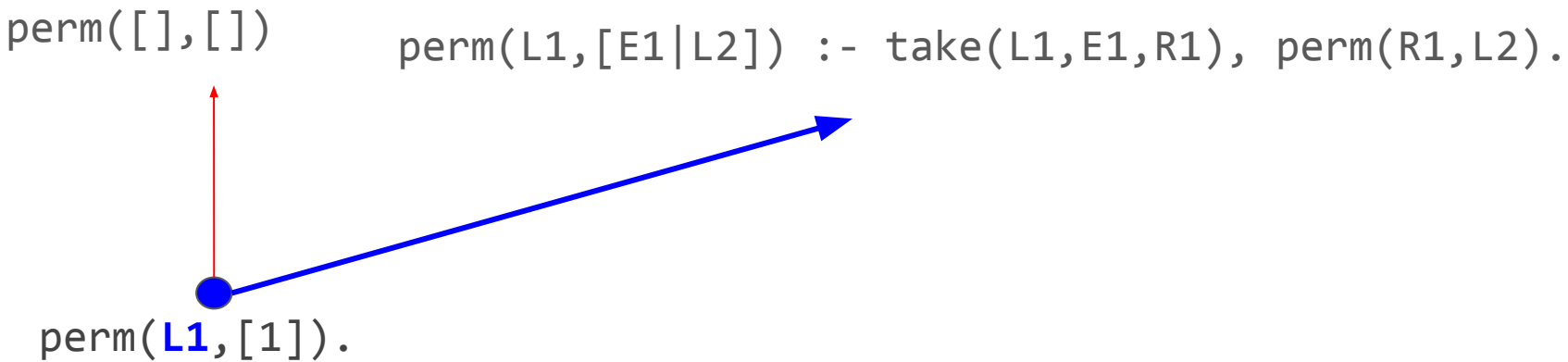
```
perm([],[])
```

```
perm(L1,[E1|L2]) :- take(L1,E1,R1), perm(R1,L2).
```

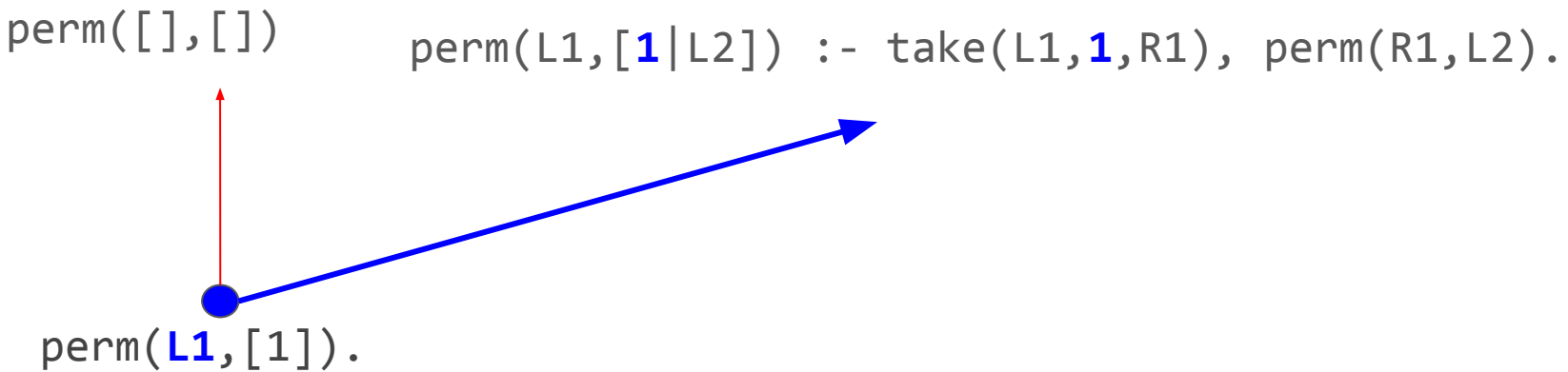


```
perm(L1,[1]).
```

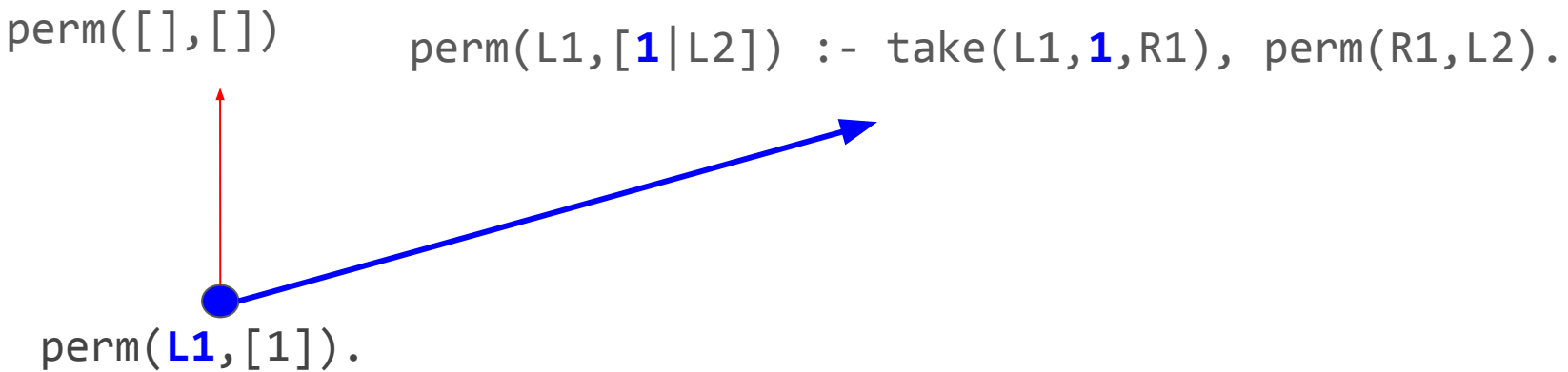
[E1|L2] = [1|[]]



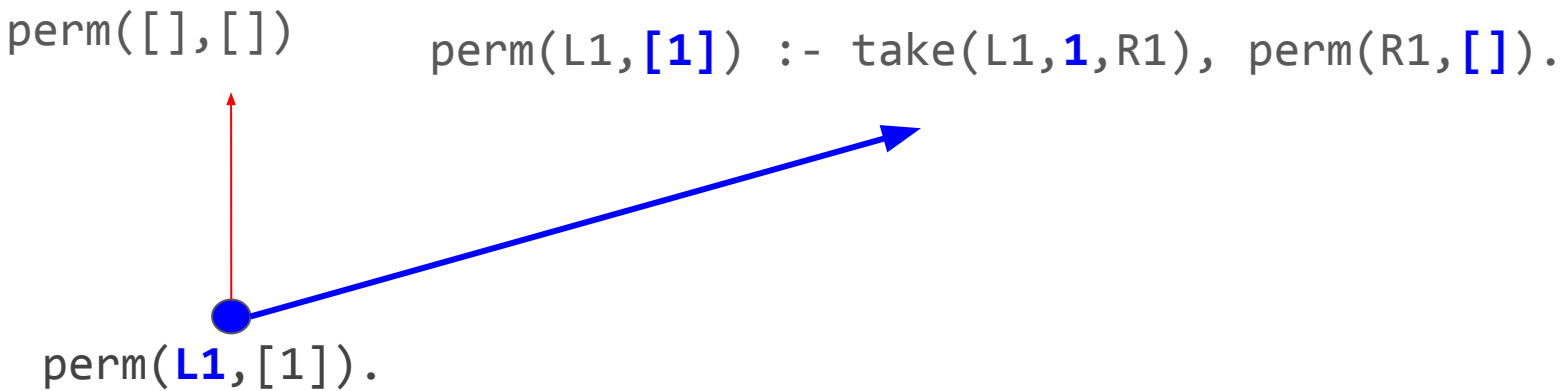
[E1|L2] = [1|[]]



[E1|L2] = [1|[]]

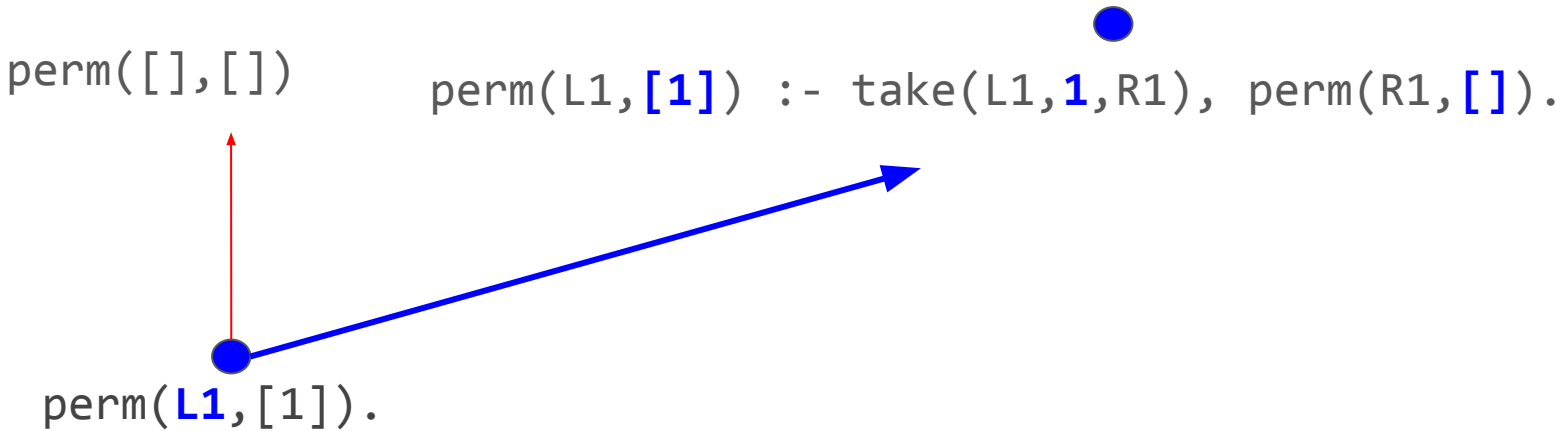


[E1|L2] = [1|[]]



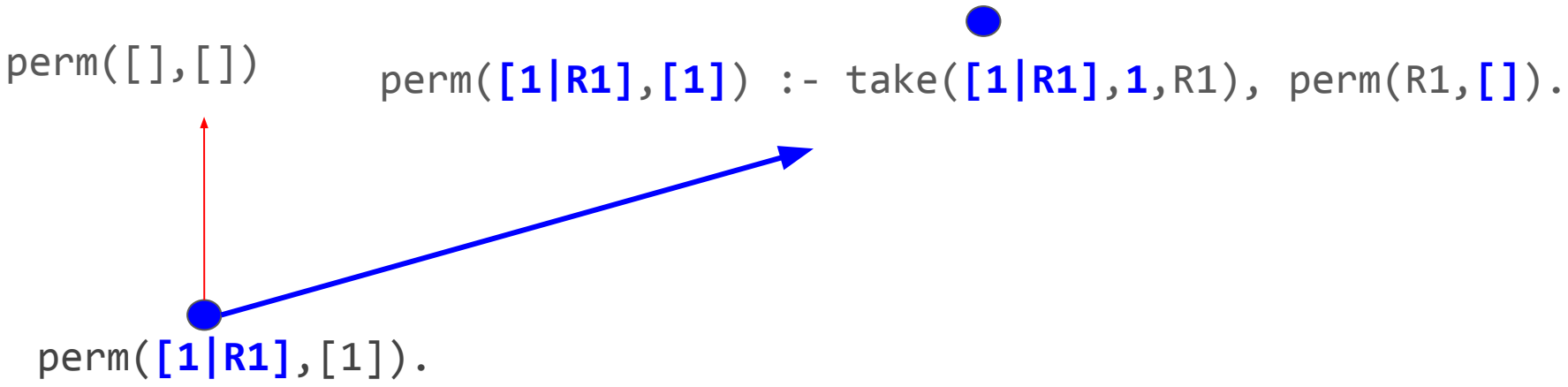
We know what take(L1,1,R1) does:

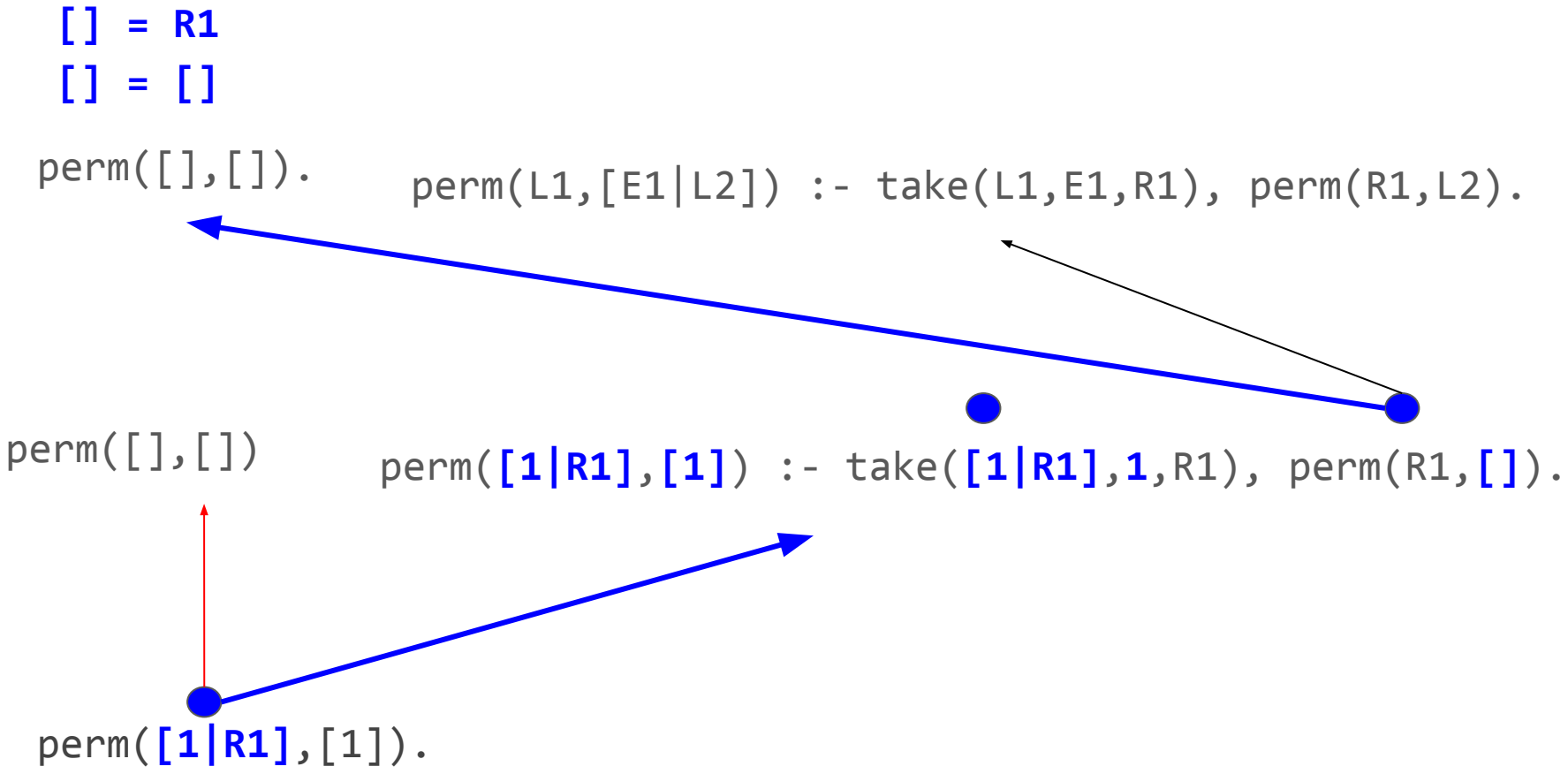
L1 = [1|R1]

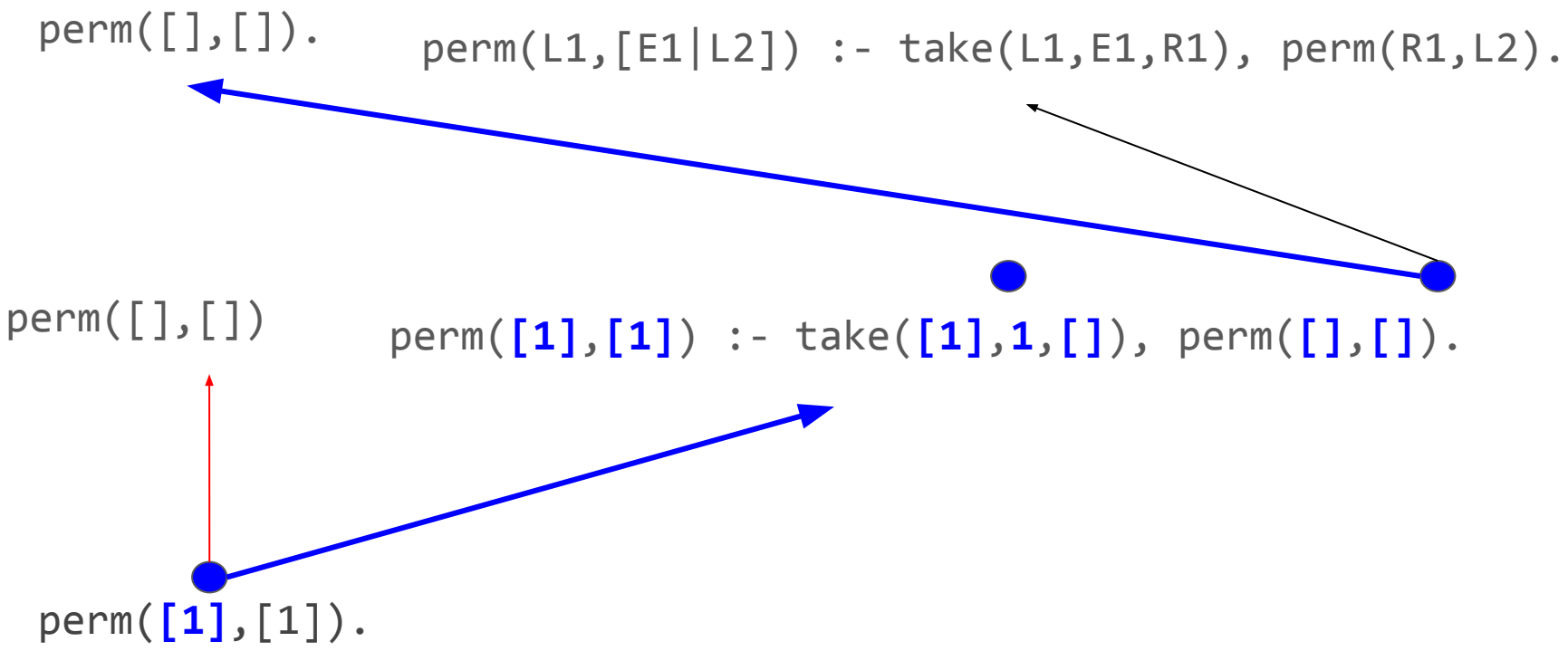


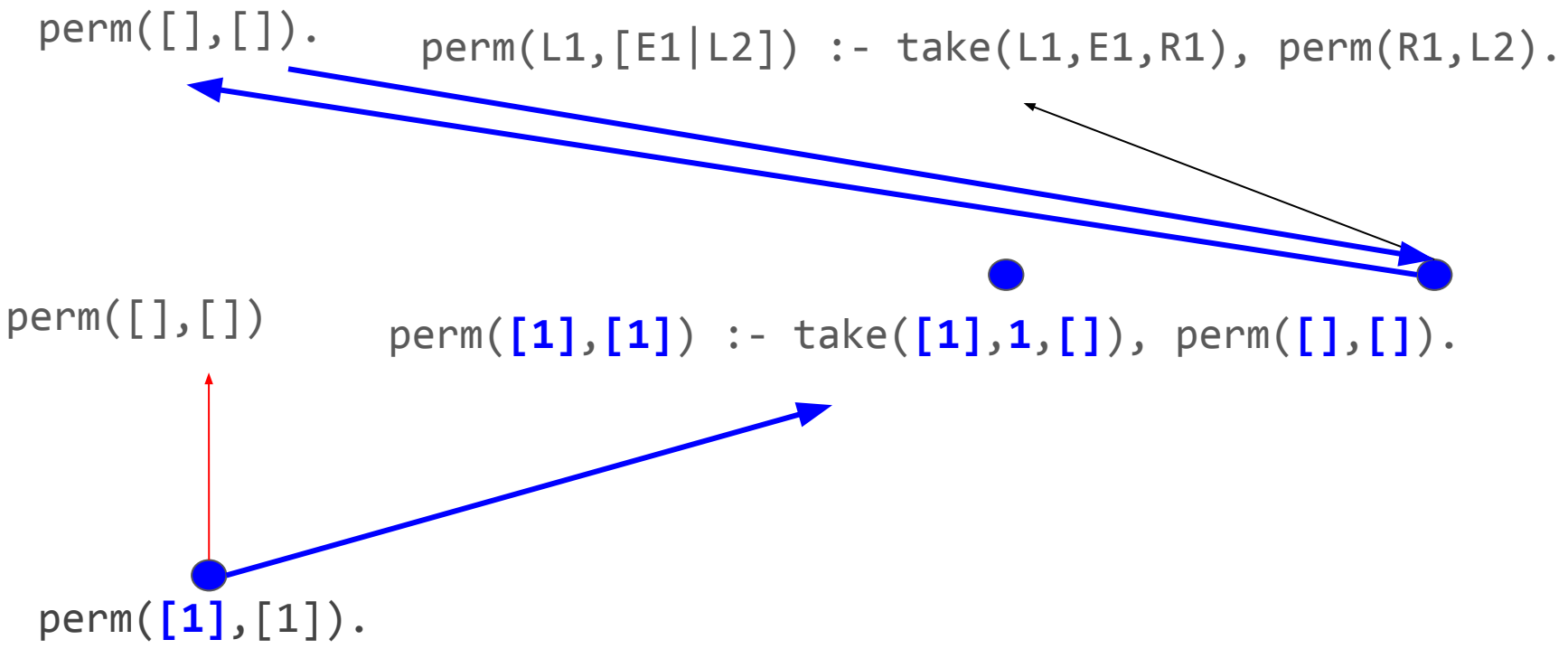
We know what take(L1,1,R1) does:

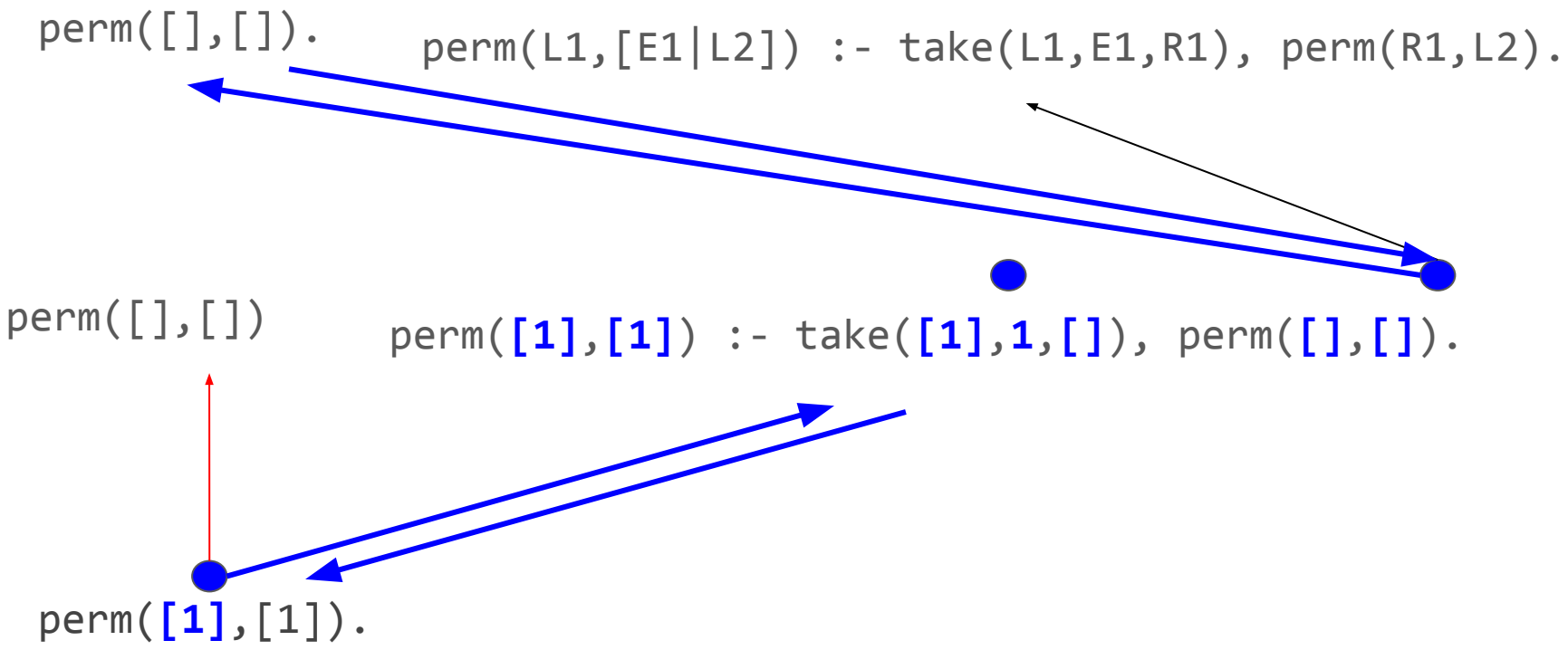
L1 = [1|R1]

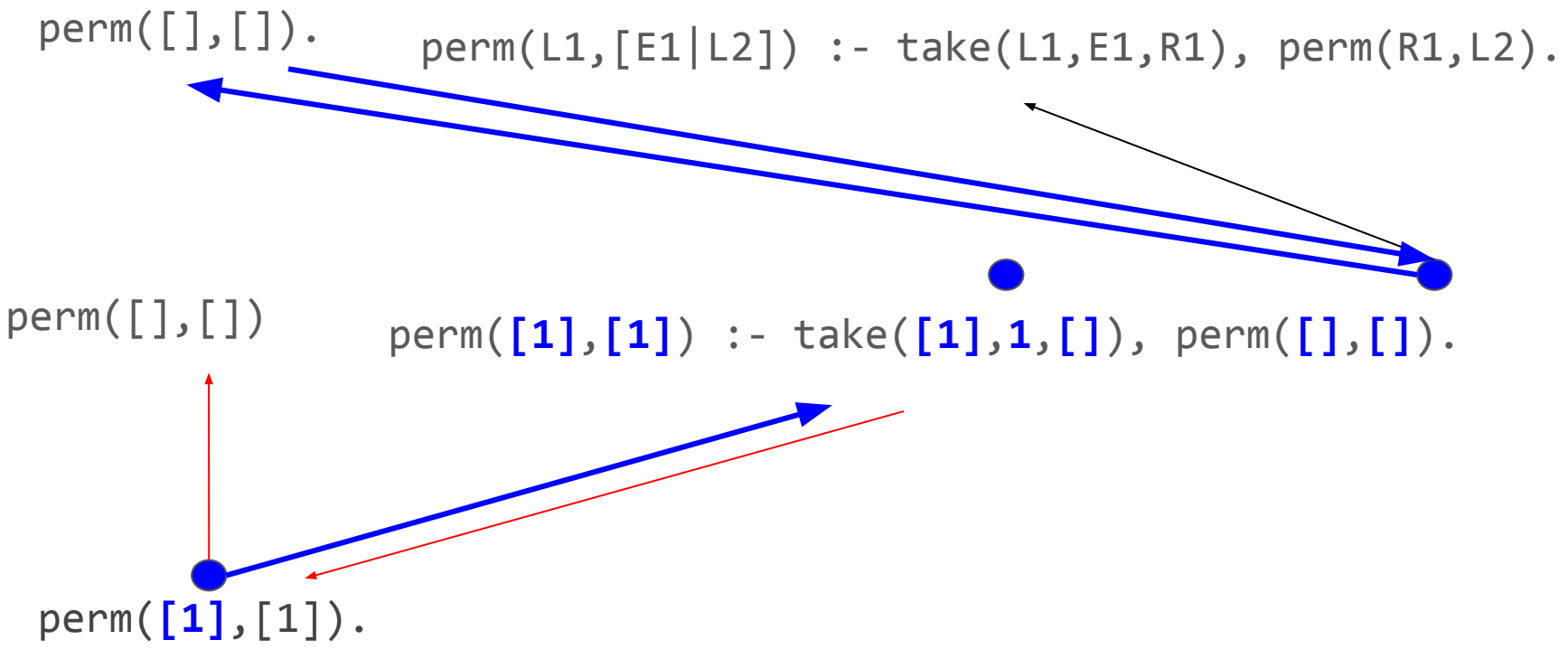


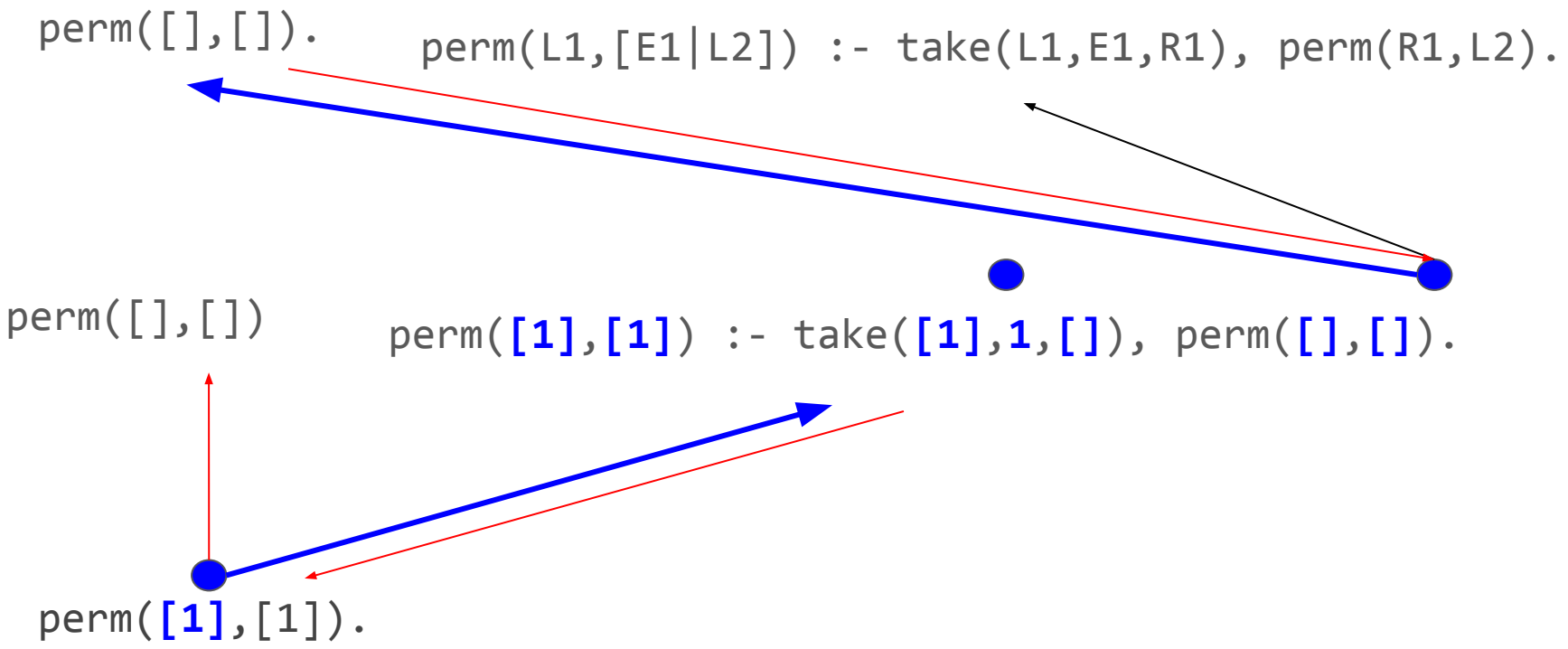

[] = R1
[] = []
perm([],[]).
perm(L1,[E1|L2]) :- take(L1,E1,R1), perm(R1,L2).
perm([],[])
perm([1|R1],[1]) :- take([1|R1],1,R1), perm(R1,[]).
perm([1|R1],[1]).

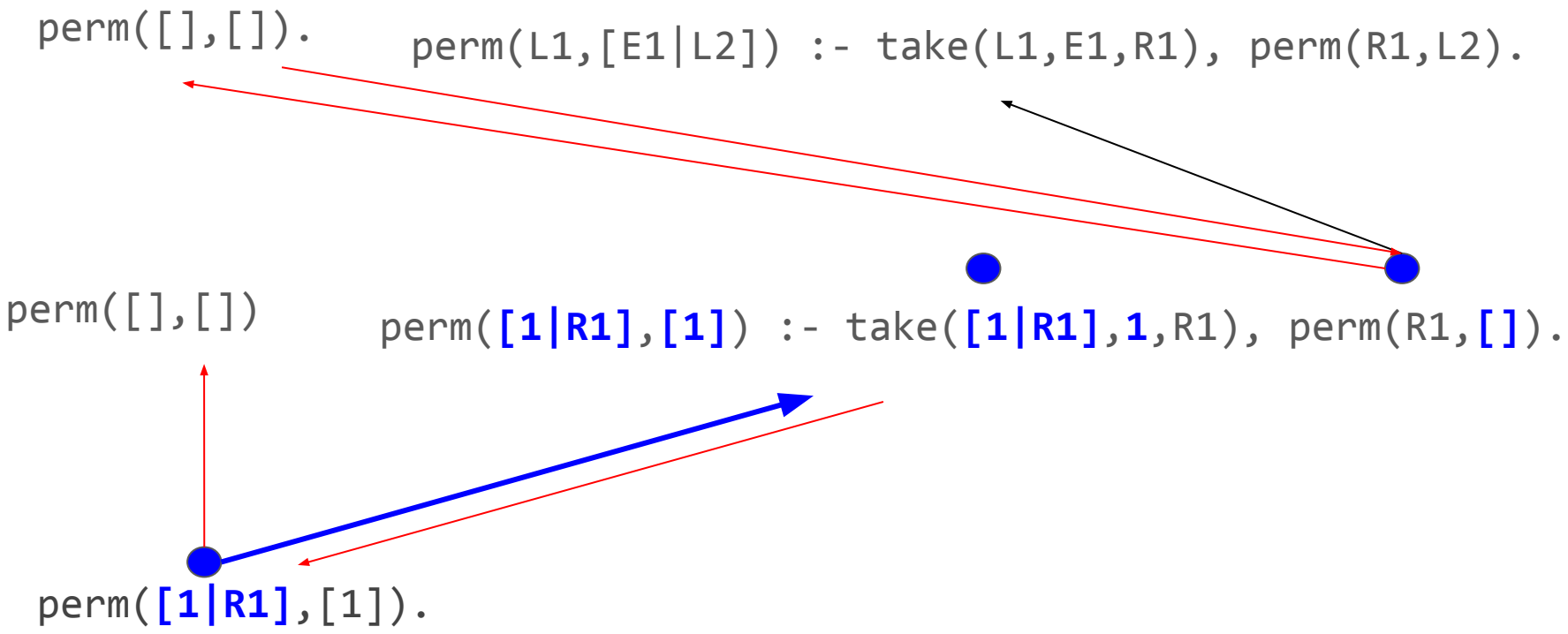

perm([],[]).
perm(L1,[E1|L2]) :- take(L1,E1,R1), perm(R1,L2).
perm([],[])
perm([1],[1]) :- take([1],1,[]), perm([],[]).
perm([1],[1]).

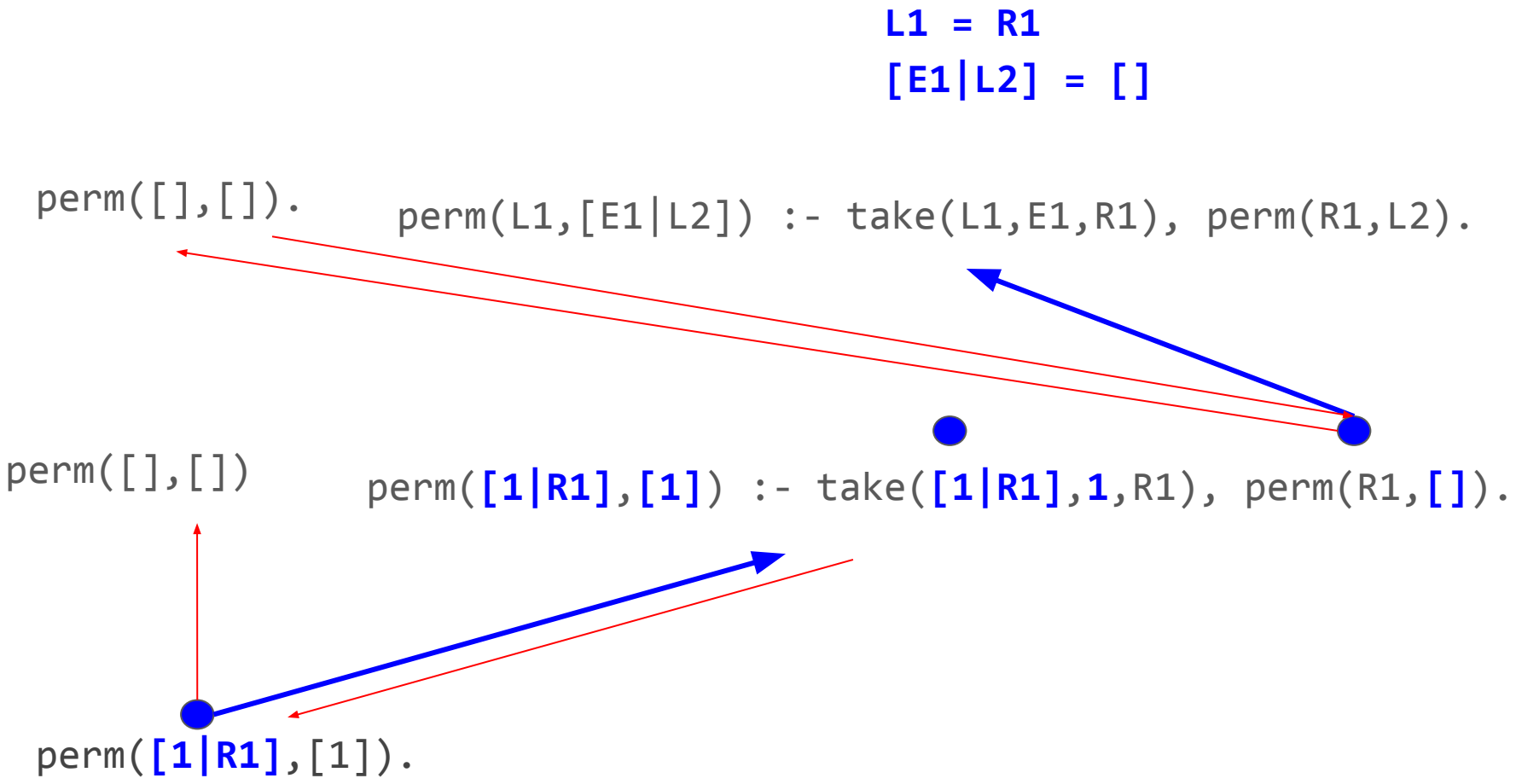

perm([],[]).
perm(L1,[E1|L2]) :- take(L1,E1,R1), perm(R1,L2).
perm([],[])
perm([1],[1]) :- take([1],1,[]), perm([],[]).
perm([1],[1]).

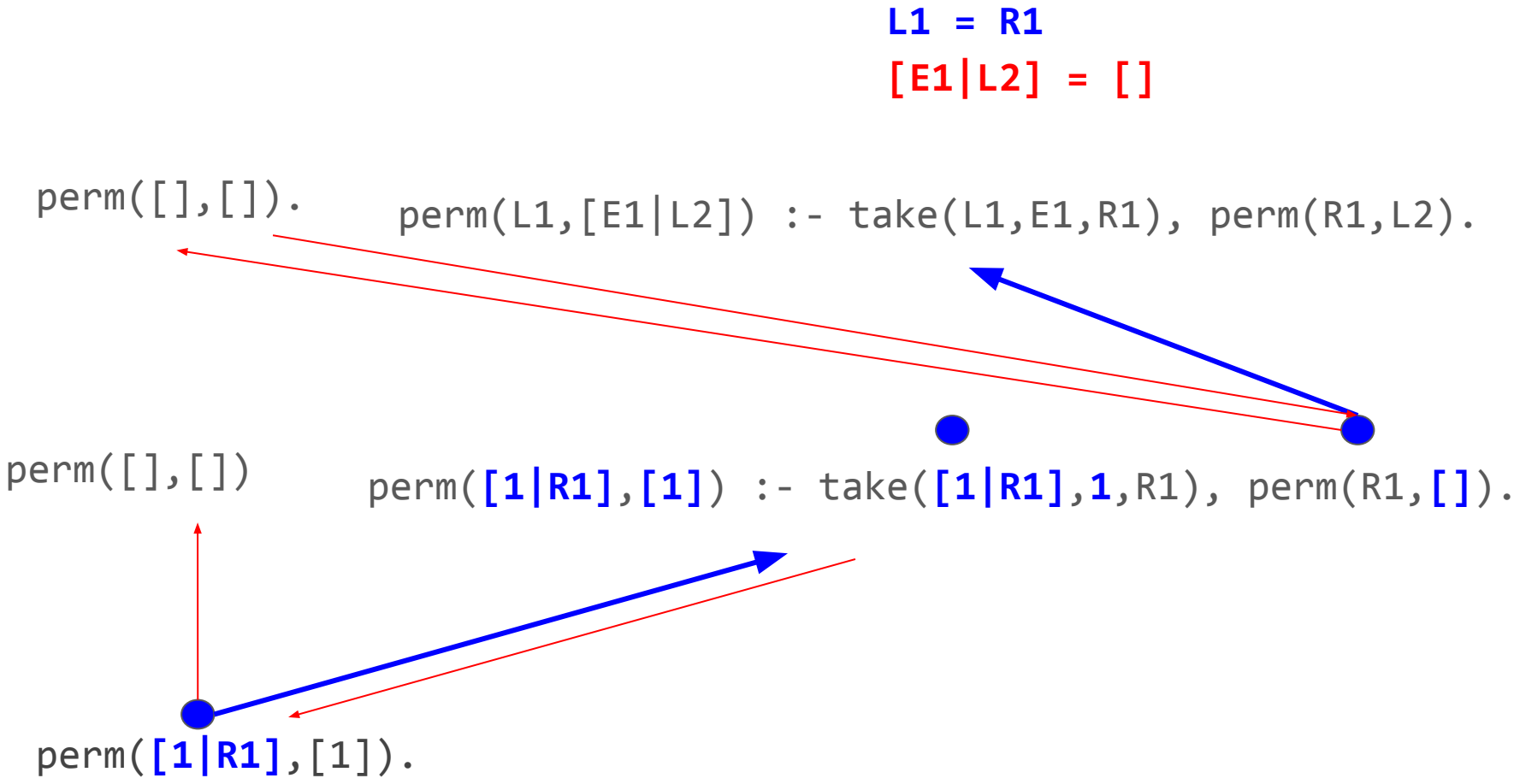

perm([],[]).
perm(L1,[E1|L2]) :- take(L1,E1,R1), perm(R1,L2).
perm([],[])
perm([1],[1]) :- take([1],1,[]), perm([],[]).
perm([1],[1]).

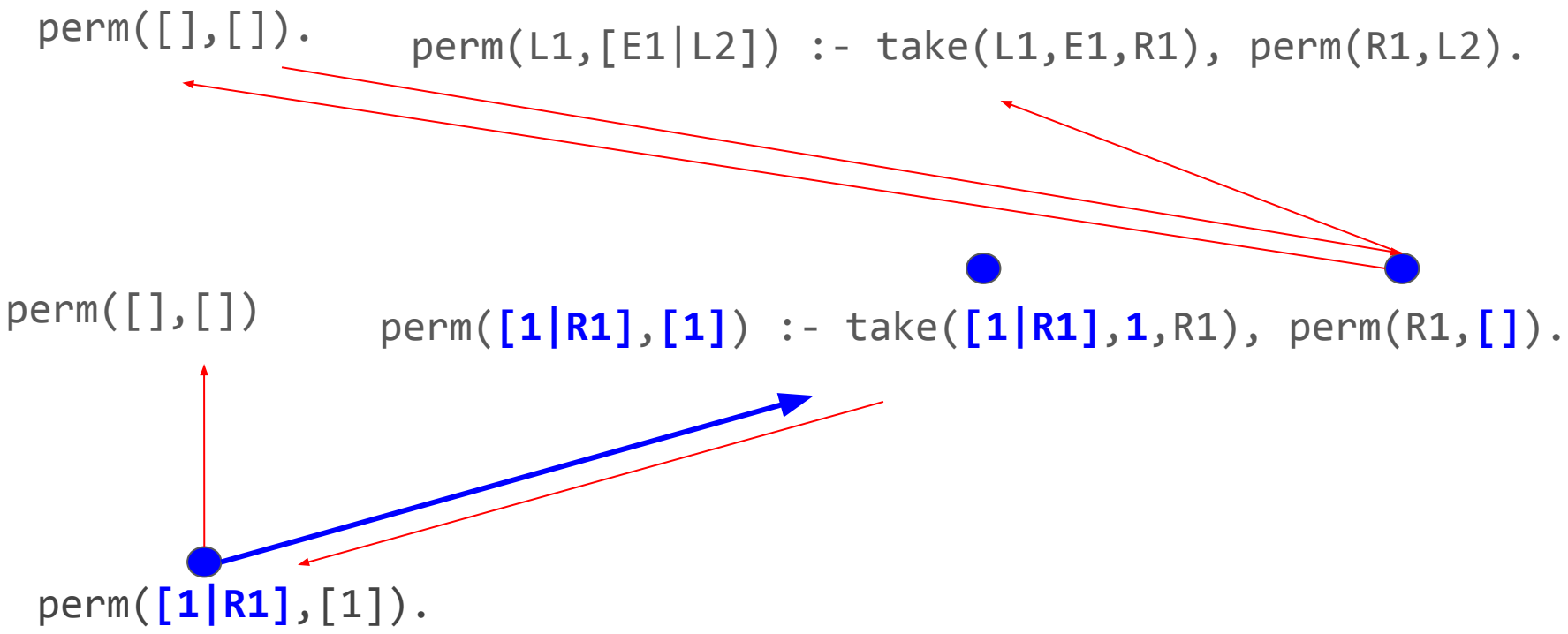

perm([],[]).
perm(L1,[E1|L2]) :- take(L1,E1,R1), perm(R1,L2).
perm([],[])
perm([1],[1]) :- take([1],1,[]), perm([],[]).
perm([1],[1]).

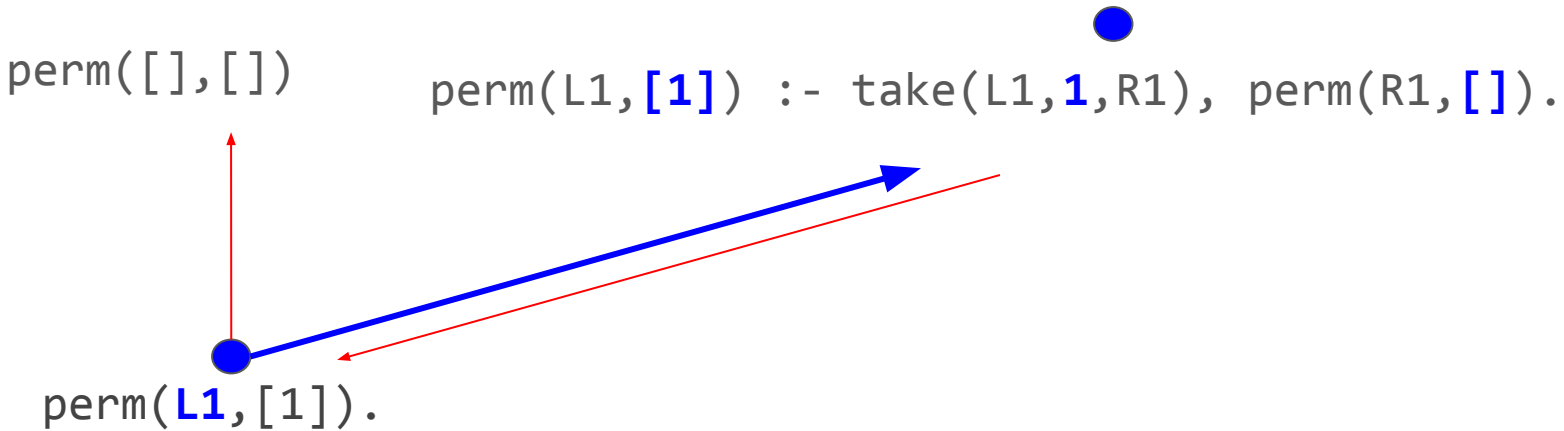

perm([],[]).
perm(L1,[E1|L2]) :- take(L1,E1,R1), perm(R1,L2).
perm([],[])
perm([1],[1]) :- take([1],1,[]), perm([],[]).
perm([1],[1]).


perm([],[]).
perm(L1,[E1|L2]) :- take(L1,E1,R1), perm(R1,L2).
perm([],[])
perm([1|R1],[1]) :- take([1|R1],1,R1), perm(R1,[]).
perm([1|R1],[1]).


L1 = R1
[E1|L2] = []
perm([],[]).
perm(L1,[E1|L2]) :- take(L1,E1,R1), perm(R1,L2).
perm([],[])
perm([1|R1],[1]) :- take([1|R1],1,R1), perm(R1,[]).
perm([1|R1],[1]).

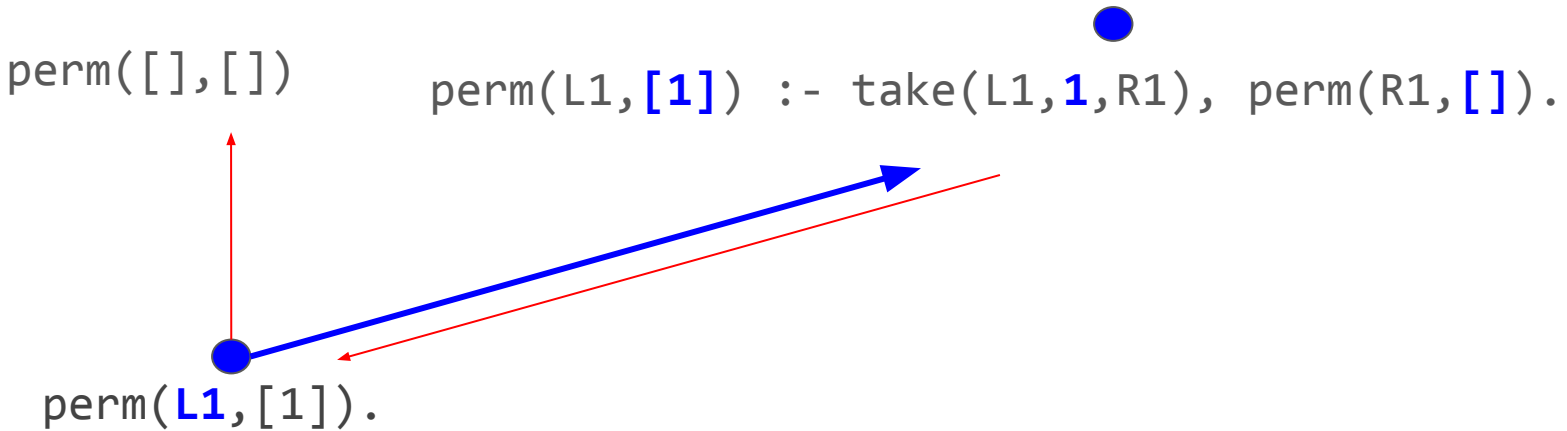

L1 = R1
[E1|L2] = []
perm([],[]).
perm(L1,[E1|L2]) :- take(L1,E1,R1), perm(R1,L2).
perm([],[])
perm([1|R1],[1]) :- take([1|R1],1,R1), perm(R1,[]).
perm([1|R1],[1]).


perm([],[]).
perm(L1,[E1|L2]) :- take(L1,E1,R1), perm(R1,L2).
perm([],[])
perm([1|R1],[1]) :- take([1|R1],1,R1), perm(R1,[]).
perm([1|R1],[1]).


perm([],[])
perm(L1,[1]) :- take(L1,1,R1), perm(R1,[]).
perm(L1,[1]).

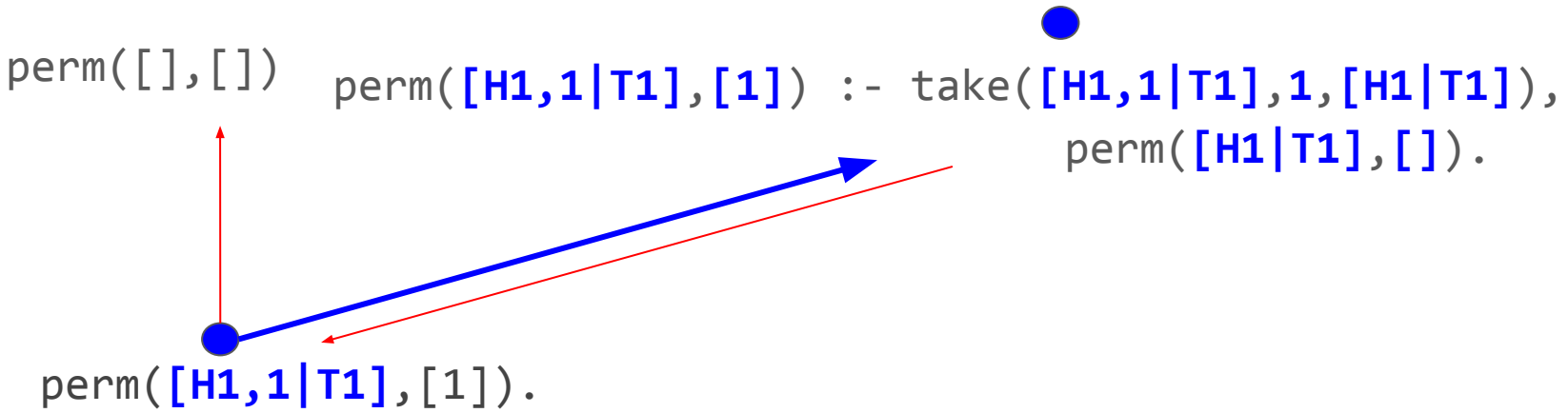
We know what backtracking take(L1,1,R1) does:

L1 = [H1,1|T1], R1 = [H1|T1]

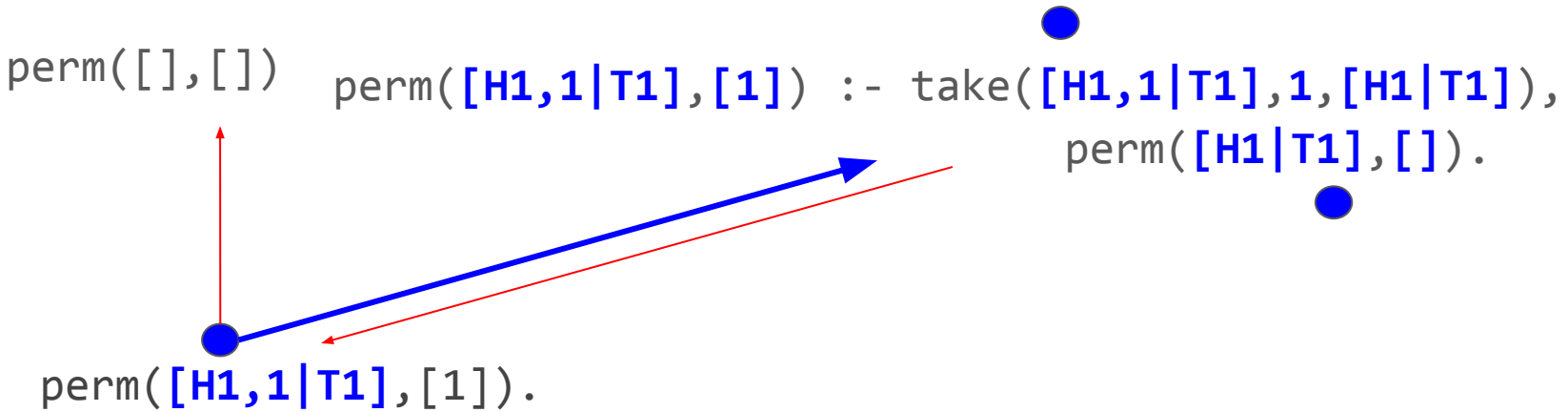


We know what backtracking take(L1,1,R1) does:

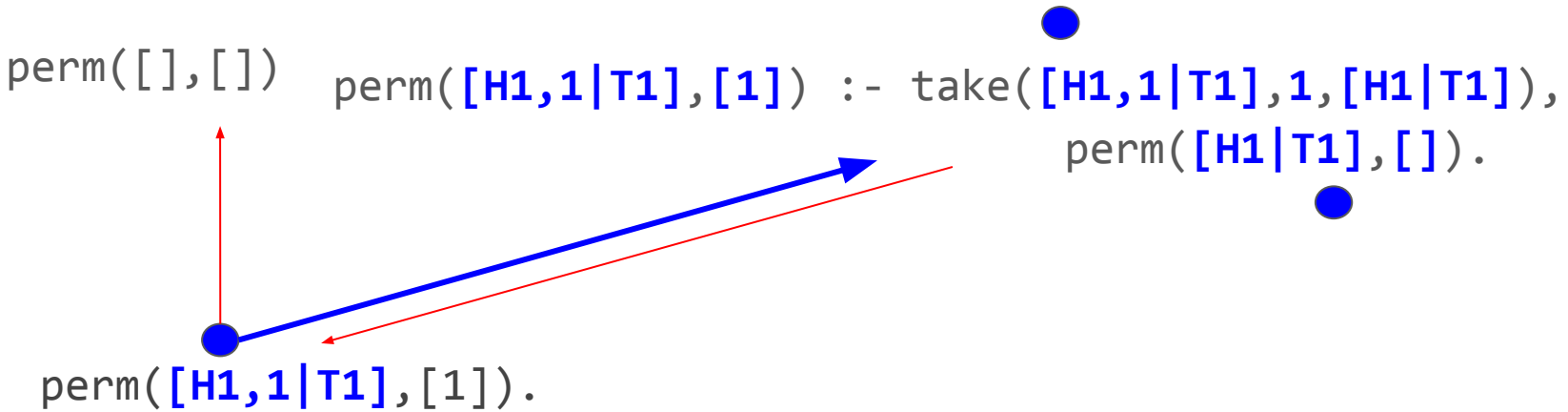
L1 = [H1,1|T1], R1 = [H1|T1]



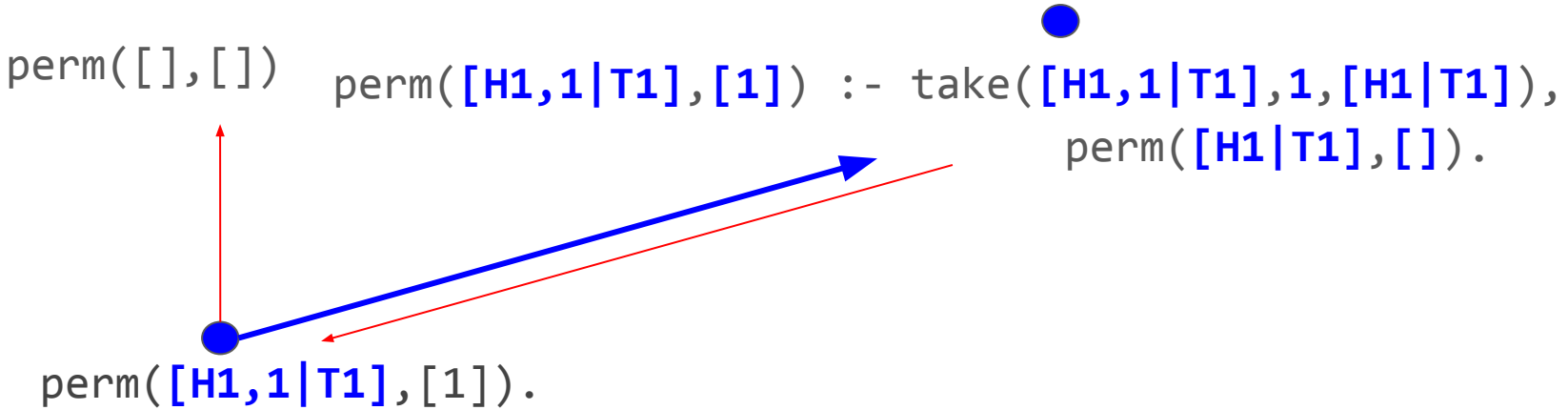
We next try perm([],[]). This fails because we would need [H1|T1] = []



We next try perm(L1,[E1|L2]). This fails because we would need [] = [E1|L2].



So we backtrack to take, which generates an even longer list L1 = [H1,H2,1|T1], R1=[H1,H2|T1]



Q: I often write logically-correct code which doesn't terminate. What heuristics can I apply to see if this will happen without running the code?

A: sometimes it can be really hard to see what's going on… :-(

Challenge: fix it - implement perm(?L1,?L2)

1) I'm done
2) I need a hint
3) You totally lost me with all those trees!
4) and I give up

http://etc.ch/8WDC