Aula 6: Mais Listas

- Teoria
  - Definir **concatena/3**, um predicado para concatenar duas listas e ilustrar o que pode ser feito com ele.
  - Discutir dois modos de **inverter** uma lista
    - Um modo simplório usando concatena/3
    - Um método mais eficiente usando acumuladores

Concatena

- Nós definiremos um importante predicado **concatena/3** cujos argumentos são todos listas.
- Declarativamente, concatena(L1,L2,L3) é verdadeiro se a lista L3 é o resultado da concatenação das listas L1 e L2

```
?- concatena([a,b,c,d],[3,4,5],[a,b,c,d,3,4,5]).
true

?- concatena([a,b,c],[3,4,5],[a,b,c,d,3,4,5]).
false
```

Concatena visto proceduralmente

- De uma perspectiva procedural, o uso mais óbvio de **concatena/3** é concatenar duas listas.
- Podemos fazer isto simplesmente usando uma variável como o terceiro argumento

```
?- concatena([a,b,c,d],[1,2,3,4,5], X).
```

Concatena visto proceduralmente

- De uma perspectiva procedural, o uso mais óbvio de **concatena/3** é concatenar duas listas.
- Podemos fazer isto simplesmente usando uma variável como o terceiro argumento

```
?- concatena([a,b,c,d],[1,2,3,4,5], X).
X=[a,b,c,d,1,2,3,4,5]
true

?-
```

Definição de concatena/3

```
concatena([], L, L).
concatena([H|L1], L2, [H|L3]):-
    concatena(L1, L2, L3).
```

- Definição recursiva
  - Cláusula base: concatenar a lista vazia com alguma outra lista produz a mesma lista.
  - O passo recursivo enuncia que ao concatenar uma lista não vazia [H|T] com uma lista L, o resultado é uma lista cuja cabeça é H e cuja cauda é o resultado da concatenação de T com L.

Como concatena/3 funciona

- Duas formas de responder:
  - Usar trace/0 com alguns exemplos
  - Desenhar uma árvore de busca!
    Vamos considerar um exemplo simples

```
?- concatena([a,b,c],[1,2,3], R).
```

Árvore de busca do exemplo

?- concatena([a,b,c],[1,2,3], R).

```
concatena([], L, L).
concatena([H|L1], L2, [H|L3]):-
    concatena(L1, L2, L3).
```

Árvore de busca do exemplo

?- concatena([a,b,c],[1,2,3], R).

/ \

```
concatena([], L, L).
concatena([H|L1], L2, [H|L3]):-
    concatena(L1, L2, L3).
```

Árvore de busca do exemplo

```
concatena([], L, L).
concatena([H|L1], L2, [H|L3]):-
    concatena(L1, L2, L3).
```

?- concatena([a,b,c],[1,2,3], R).

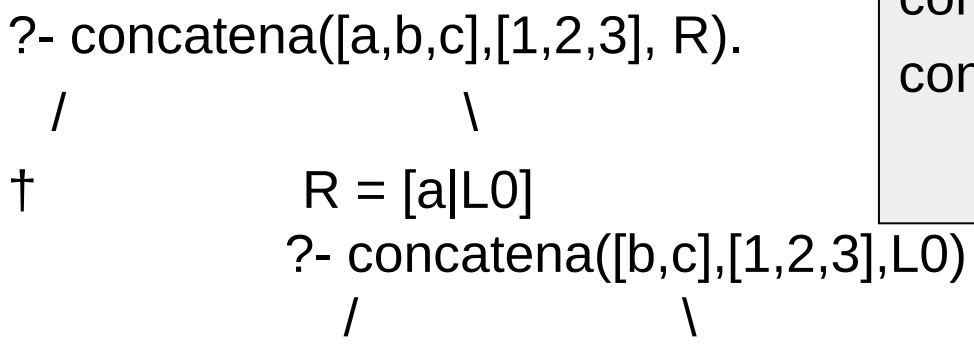
/ \

† R = [a|L0]

?- concatena([b,c],[1,2,3],L0)

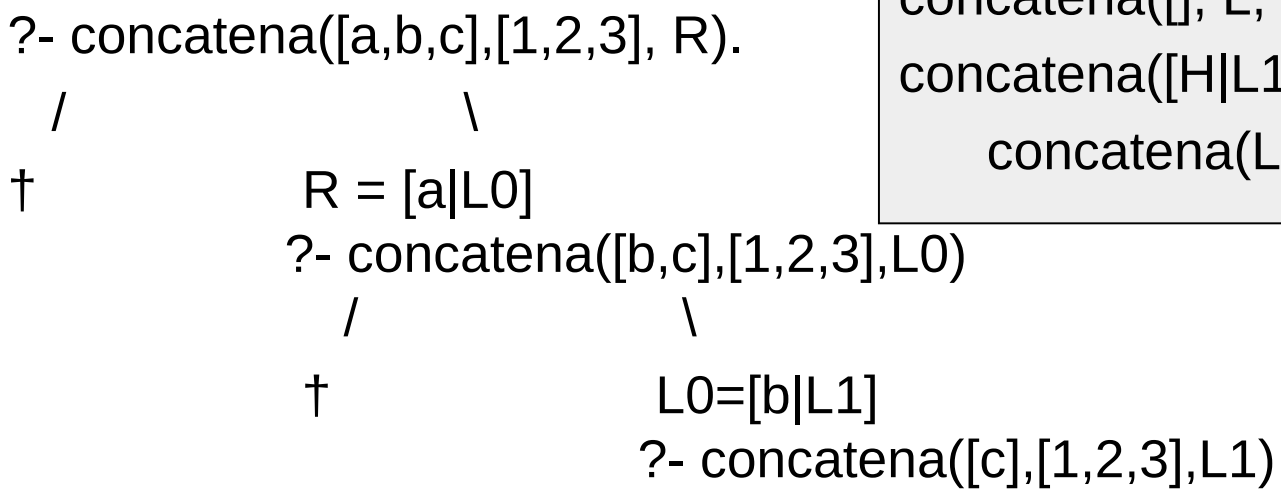
Árvore de busca do exemplo

```
concatena([], L, L).
concatena([H|L1], L2, [H|L3]):-
    concatena(L1, L2, L3).
```



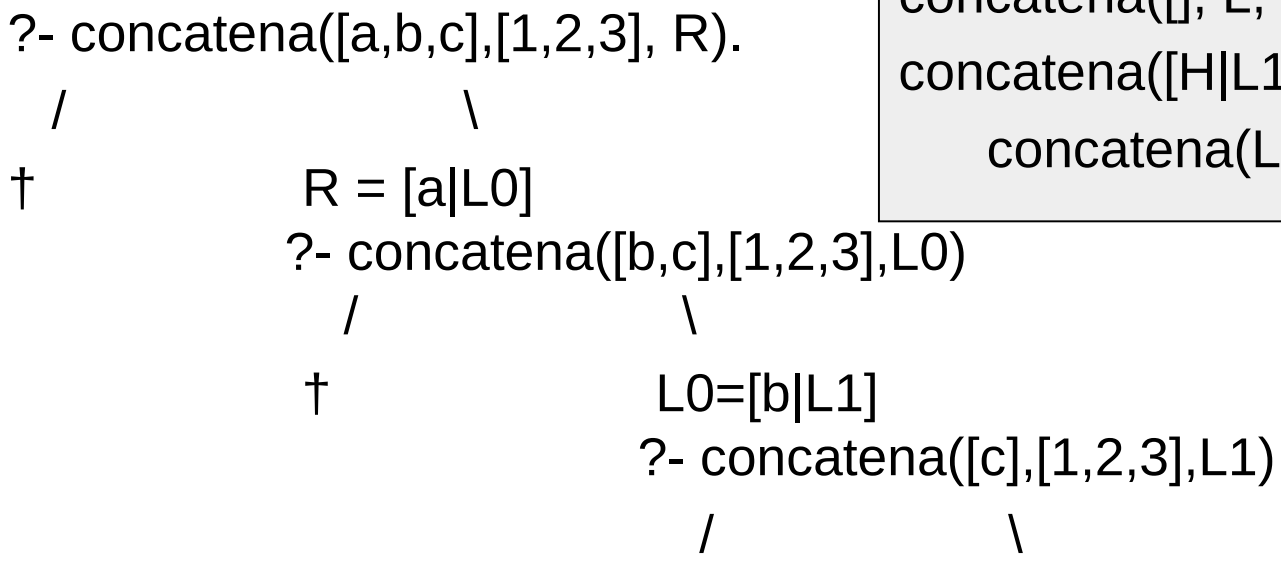
Árvore de busca do exemplo

```
concatena([], L, L).
concatena([H|L1], L2, [H|L3]):-
    concatena(L1, L2, L3).
```



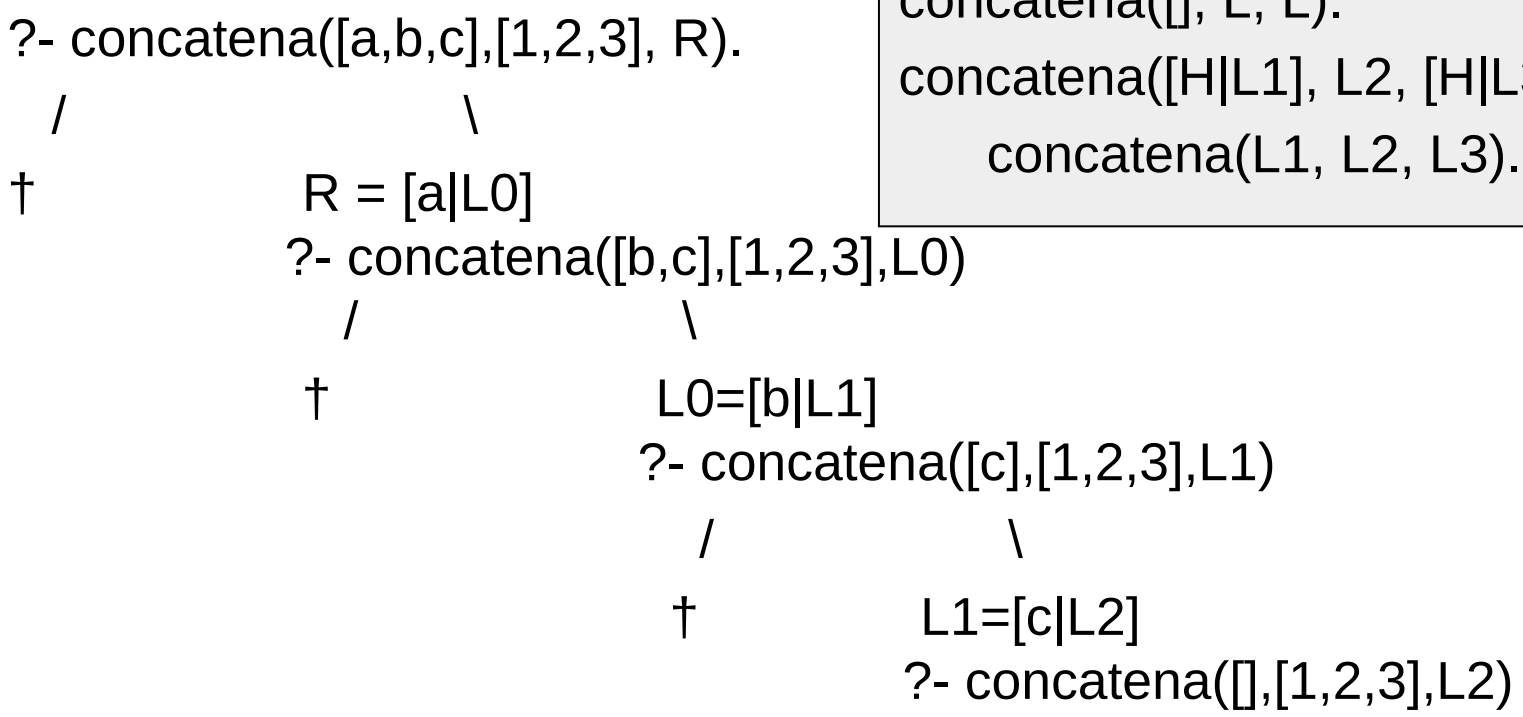
Árvore de busca do exemplo

```
concatena([], L, L).
concatena([H|L1], L2, [H|L3]):-
    concatena(L1, L2, L3).
```



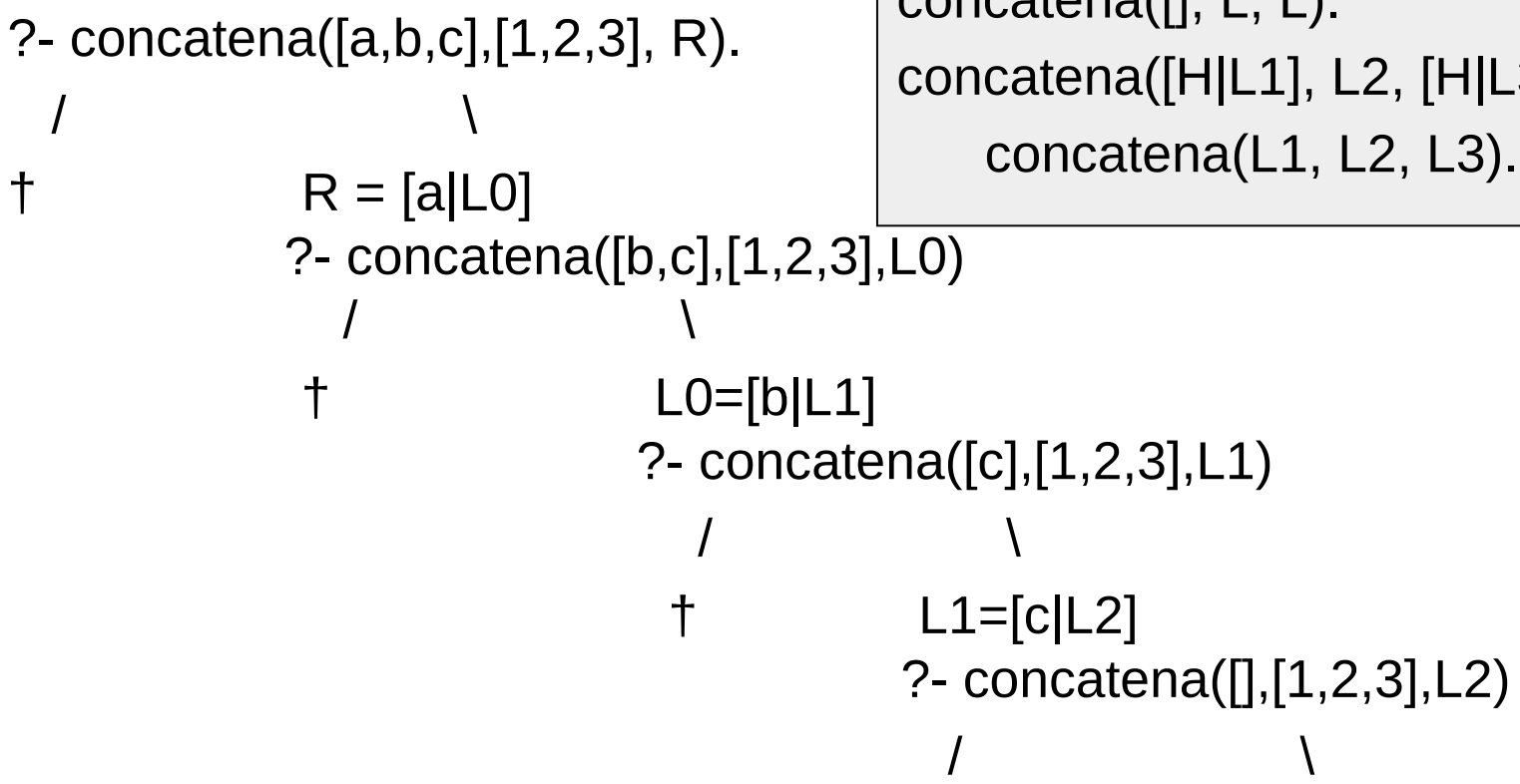
Árvore de busca do exemplo

```
concatena([], L, L).
concatena([H|L1], L2, [H|L3]):-
    concatena(L1, L2, L3).
```



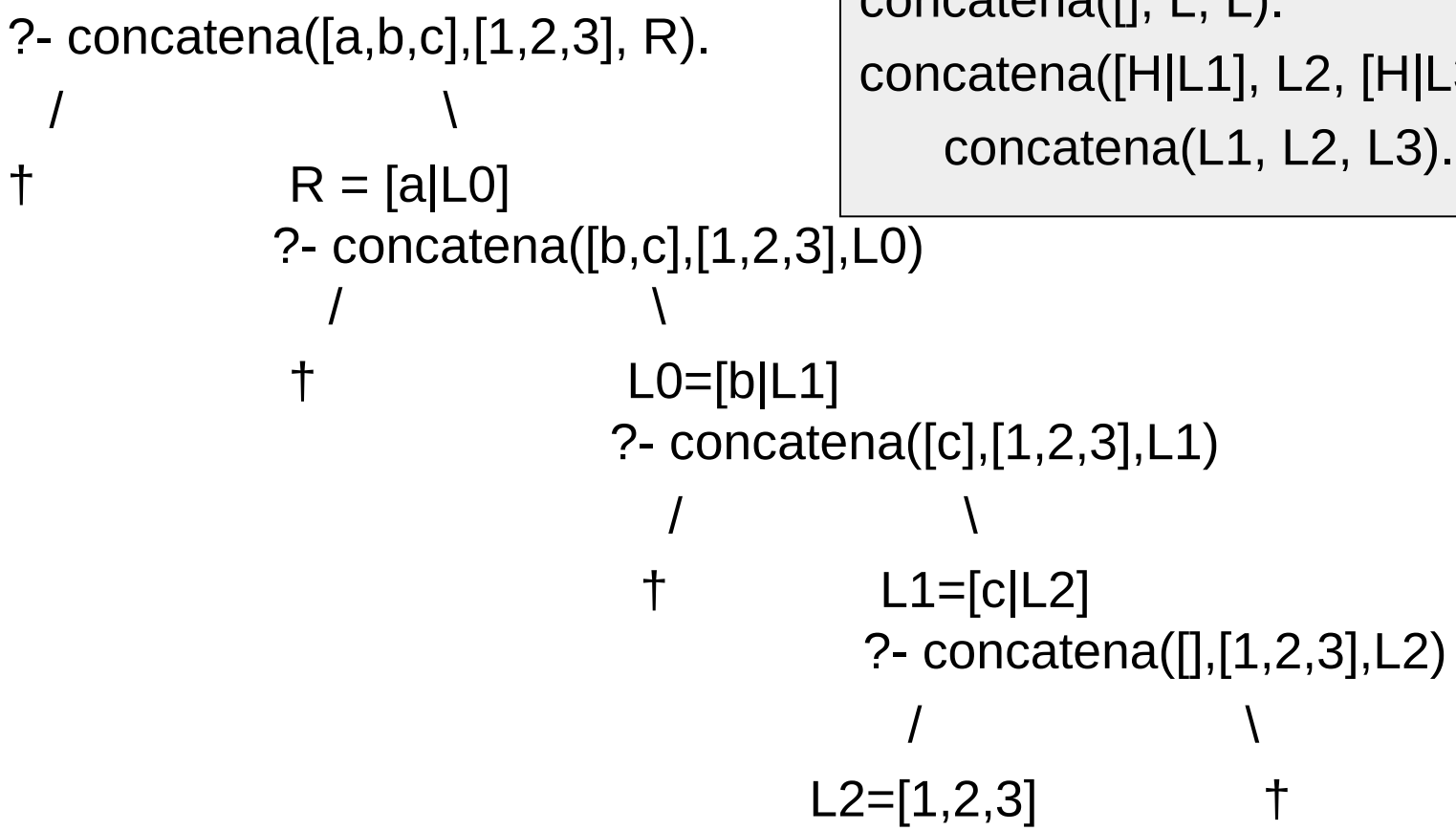
Árvore de busca do exemplo

```
concatena([], L, L).
concatena([H|L1], L2, [H|L3]):-
    concatena(L1, L2, L3).
```



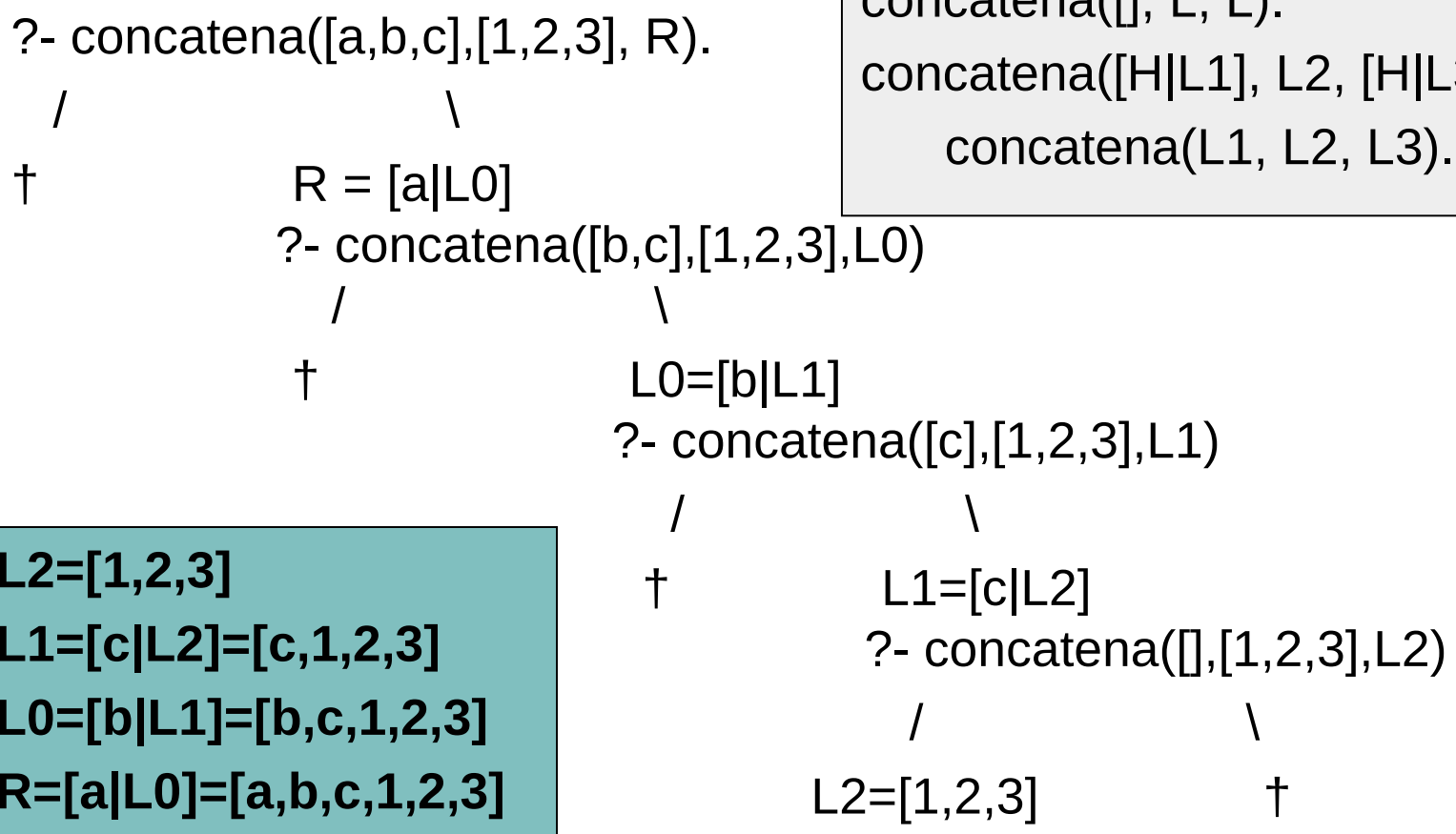
Árvore de busca do exemplo

```
concatena([], L, L).
concatena([H|L1], L2, [H|L3]):-
    concatena(L1, L2, L3).
```



Árvore de busca do exemplo

```
concatena([], L, L).
concatena([H|L1], L2, [H|L3]):-
    concatena(L1, L2, L3).
```



L2=[1,2,3]
L1=[c|L2]=[c,1,2,3]
L0=[b|L1]=[b,c,1,2,3]
R=[a|L0]=[a,b,c,1,2,3]

Usando concatena/3

- Agora que sabemos como concatena/3 funciona, vamos observar algumas aplicações
- Dividir uma lista:

```
?- concatena(X,Y, [a,b,c,d]).
X=[ ]          Y=[a,b,c,d];
X=[a]          Y=[b,c,d];
X=[a,b]        Y=[c,d];
X=[a,b,c]      Y=[d];
X=[a,b,c,d]    Y=[ ];
false
```

Prefixo e sufixo

- Podemos também usar concatena/3 para definir outros predicados úteis.
- Um bom exemplo é encontrar os prefixos e sufixos de uma lista

Definição de prefixo/2

```
prefixo(P,L):-
    concatena(P,_,L).
```

- Uma lista P é um prefixo de alguma lista L, se existir alguma lista tal que L é o resultado de concatenar P com esta lista.
- Podemos usar a variável anônima pois não nos importamos com o que esta lista seja.

Uso de prefixo/2

```
prefixo(P,L):-
    concatena(P,_,L).
```

```
?- prefixo(X, [a,b,c,d]).
X=[ ];
```

Uso de prefixo/2

```
prefixo(P,L):-
    concatena(P,_,L).
```

```
?- prefixo(X, [a,b,c,d]).
X=[ ];
X=[a];
```

Uso de prefixo/2

```
prefixo(P,L):-
    concatena(P,_,L).
```

```
?- prefixo(X, [a,b,c,d]).
X=[ ];
X=[a];
X=[a,b];
```

Uso de prefixo/2

```
prefixo(P,L):-
    concatena(P,_,L).
```

```
?- prefixo(X, [a,b,c,d]).
X=[ ];
X=[a];
X=[a,b];
X=[a,b,c];
```

Uso de prefixo/2

```
prefixo(P,L):-
    concatena(P,_,L).
```

```
?- prefixo(X, [a,b,c,d]).
X=[ ];
X=[a];
X=[a,b];
X=[a,b,c];
X=[a,b,c,d];
```

Uso de prefixo/2

```
prefixo(P,L):-
    concatena(P,_,L).
```

```
?- prefixo(X, [a,b,c,d]).
X=[ ];
X=[a];
X=[a,b];
X=[a,b,c];
X=[a,b,c,d];
false
```

Definição de sufixo/2

```
sufixo(S,L):-
    concatena(_,S,L).
```

- Uma lista S é um sufixo de alguma lista L se existir alguma lista tal que L é o resultado de concatenar esta lista com S.
- Mais uma vez usamos a variável anônima pois não nos importamos com o que esta lista seja.

Uso de sufixo/2

```
sufixo(S,L):-
    concatena(_,S,L).
```

```
?- sufixo(X, [a,b,c,d]).
X=[a,b,c,d];
```

Uso de sufixo/2

```
sufixo(S,L):-
    concatena(_,S,L).
```

```
?- sufixo(X, [a,b,c,d]).
X=[a,b,c,d];
X=[b,c,d];
```

Uso de sufixo/2

```
sufixo(S,L):-
    concatena(_,S,L).
```

```
?- sufixo(X, [a,b,c,d]).
X=[a,b,c,d];
X=[b,c,d];
X=[c,d];
```

Uso de sufixo/2

```
sufixo(S,L):-
    concatena(_,S,L).
```

```
?- sufixo(X, [a,b,c,d]).
X=[a,b,c,d];
X=[b,c,d];
X=[c,d];
X=[d];
```

Uso de sufixo/2

```
sufixo(S,L):-
    concatena(_,S,L).
```

```
?- sufixo(X, [a,b,c,d]).
X=[a,b,c,d];
X=[b,c,d];
X=[c,d];
X=[d];
X=[];
```

Uso de sufixo/2

```
sufixo(S,L):-
    concatena(_,S,L).
```

```
?- sufixo(X, [a,b,c,d]).
X=[a,b,c,d];
X=[b,c,d];
X=[c,d];
X=[d];
X=[];
false
```

Definição de sublista/2

- Agora é muito fácil escrever um predicado que encontra sublistas de uma lista.
- As sublistas de uma lista L são os prefixos dos sufixos de L.

```
sublista(Sub,Lista):-
    sufixo(Sufixo,Lista),
    prefixo(Sub,Sufixo).
```

concatena/3 e eficiência

- O predicado **concatena/3** é útil e é importante saber como usá-lo.
- É de igual importância saber que **concatena/3** pode ser uma fonte de ineficiência
- Por que?
  - A concatenação de uma lista não é realizada com uma única ação.
  - Mas, pela travessia de uma das listas.

Pergunta

- Usando **concatena/3** nós gostaríamos de concatenar duas listas:
  - Lista 1: [a,b,c,d,e,f,g,h,i]
  - Lista 2: [j,k,l]
- O resultado deveria ser uma lista com todos os elementos das listas 1 e 2, a ordem dos elementos não é importante.
- Quais das metas seguintes é o modo mais eficiente de concatenar duas listas?

  ?- concatena([a,b,c,d,e,f,g,h,i],[j,k,l],R).

  ?- concatena([j,k,l],[a,b,c,d,e,f,g,h,i],R).

Resposta

- Observe o modo com o qual **concatena/3** é definido.
- Ele recorre sobre o primeiro argumento, sem tocar no segundo argumento.
- Isto significa que é melhor chamá-lo com a lista mais curta como o primeiro argumento.
- Naturalmente, você nem sempre sabe qual é a lista mais curta e somente pode fazer isto se não se importar com a ordem dos elementos na lista concatenada.
- Porém, se você pode fazer isto, o seu código Prolog poderá ser mais eficiente.

Exercícios

- Chamaremos uma lista de **duplicada** se ela é formada de dois blocos consecutivos de elementos que são exatamente os mesmos.
- Por exemplo, **[a,b,c,a,b,c]** é duplicada, pois ela é formada de **[a,b,c]** seguida por **[a,b,c]**.
- Também é duplicada a lista **[fu,ba,fu,ba]**.
- Por outro lado, a lista **[fu,ba,fu]** não é duplicada.
- Escreva um predicado `duplicada(Lista)` que é verdadeiro quando `Lista` é uma lista duplicada.

Exercícios

Um palíndromo é uma palavra ou frase que tenha a propriedade de poder ser lida tanto da direita para a esquerda quanto da esquerda para a direita da mesma forma. Por exemplo, “rodador", “ama" e “anilina" são palíndromos.

Escreva um predicado `palindromo(Lista)` que verifica se `Lista` é um palíndromo.

Alguns exemplos de consultas,

?- palindromo([r,o,d,a,d,o,r]).

true

?- palindromo([a,d,r,o,g,a,d,a,g,o,r,d,a]).

true

?- palindromo([e,s,s,e,n,a,o]).

false

Invertendo uma Lista

- Nós ilustraremos o problema com concatena/3, usando-o para inverter os elementos de uma lista.
- Isto é, nós definiremos um predicado que muda uma lista [a,b,c,d,e] para a lista [e,d,c,b,a]
- Isto seria uma ferramenta útil de se ter, pois o Prolog somente permite acessar facilmente os elementos na frente da lista.

Inversão simplória

- Definição recursiva
  - Se invertemos a lista vazia, obtemos a lista vazia.
  - Se invertemos a lista [H|T], nós terminamos com a lista obtida pela inversão de T e concatenada com [H]
- Para se certificar que esta definição é correta, considere a lista [a,b,c,d].
  - Se invertemos a cauda da lista, ficamos com [d,c,b].
  - Concatenando isto com [a] produz [d,c,b,a]

Inversão simplória em Prolog

```
inverteSimp([],[]).
inverteSimp([H|T],R):-
    inverteSimp(T,RT),
    concatena(RT,[H],R).
```

- Esta definição é correta, mas gasta uma grande quantidade de trabalho.
- Ela gasta a maior parte do tempo realizando concatenações.
- Entretanto, existe um modo melhor…

Inversão usando um acumulador

- O modo melhor é usar um acumulador
- O acumulador será uma lista, e no início da inversão ela estará vazia.
- Nós simplesmente pegamos a cabeça da lista que queremos inverter e a colocamos na cabeça da lista acumuladora.
- Continuamos com isto até encontrar a lista vazia.
- Neste ponto o acumulador conterá a lista invertida!

Inversão usando um acumulador

```
inverteAcum([ ],L,L).
inverteAcum([H|T],Acum,Inv):-
    inverteAcum(T,[H|Acum],Inv).
```

Adicionando um predicado-capa

```
inverteAcum([ ],L,L).
inverteAcum([H|T],Acum,Inv):-
    inverteAcum(T,[H|Acum],Inv).
```

```
inverte(L1,L2):-
    inverteAcum(L1,[ ],L2).
```

Ilustração do acumulador

- Lista: [a,b,c,d] Acumulador: []

Ilustração do acumulador

- Lista: [a,b,c,d] Acumulador: []
- Lista: [b,c,d] Acumulador: [a]

Ilustração do acumulador

- Lista: [a,b,c,d] Acumulador: []
- Lista: [b,c,d] Acumulador: [a]
- Lista: [c,d] Acumulador: [b,a]

Ilustração do acumulador

- Lista: [a,b,c,d] Acumulador: []
- Lista: [b,c,d] Acumulador: [a]
- Lista: [c,d] Acumulador: [b,a]
- Lista: [d] Acumulador: [c,b,a]

Ilustração do acumulador

- Lista: [a,b,c,d] Acumulador: []
- Lista: [b,c,d] Acumulador: [a]
- Lista: [c,d] Acumulador: [b,a]
- Lista: [d] Acumulador: [c,b,a]
- Lista: [] Acumulador: [d,c,b,a]

Resumo desta aula

- O **concatena/3** é um predicado útil, não tenha medo de usá-lo
- Entretanto, ele pode ser uma fonte de ineficiência
- O uso de acumuladores é normalmente melhor
- Nós encontraremos um modo muito eficiente de concatenar listas nas próximas aulas, onde exploraremos o uso das “listas de diferenças”

Próxima aula

- Gramáticas de Cláusulas definidas
  - Introdução de gramáticas livres de contexto e conceitos relacionados.
  - Introdução das DCGs, gramáticas de cláusulas definidas, um mecanismo já existente no Prolog para trabalhar com gramáticas livres de contexto.