

Resolução de provas antigas

AP2 - 2024.2

Questão 1)

```
fun {AppendD D1 D2 }
    S1#E1=D1
    S2#E2=D2
in
    E1=S2
    S1#E2
end
```

Não é possível concatenar a primeira lista de uma aplicação do AppendD mais de uma vez devido às variáveis serem de atribuição única, isto é, ao serem ligadas a um valor ou à outra variável, esta ligação não pode ser alterada.

Suponha duas listas diferença $Xs = [a\ b\ c\ X]\#X$ e $Ys = [d\ e\ Y]\#Y$. Chamemos, então, $\{AppendD\ Xs\ Ys\}$, fazendo com que as listas Xs e Ys sejam concatenadas em tempo $O(1)$. Dessa forma, Xs , por intermédio do pattern matching, será quebrado no formato $S1#E1$, em que $S1$ representa a lista principal e $E1$ representa a variável não ligada. Análogo para Ys e $S2#E2$. Em seguida, a variável não ligada X de Xs será ligada à lista principal $[d\ e]$ de Ys , de forma que a variável em $E1$ aponte para a lista em $S2$, e Xs e Ys se tornem uma lista só. Por fim, $S1#E2$ é retornado, isto é, a lista resultante da junção de Xs e Ys em conjunto com a variável não ligada Y de Ys .

Daí, há de ressaltar que a variável não ligada da lista diferença Xs foi ligada à lista principal de Ys , isto é, X foi ligada à lista $[d\ e]$, resultando a lista $S1#E2 = [a\ b\ c\ d\ e\ Y]\#Y$. Como a variável X já foi ligada, então não é possível atribuir qualquer outro valor a ela, de forma que não se pode mais concatenar a lista Xs . Já no caso da lista Ys , a segunda lista do argumento, há ainda a variável não ligada Y , de forma que tanto Ys pode ser concatenada quanto $S1#E2$.

Questão 2)

```
fun {Append Ls Ms}
  case Ls
    of nil then Ms
    [] X|Lr then X|{Append Lr Ms}
  end
end
```

Item A)

```
proc {Append Xs Ys ?Zs}
  case Xs
    of nil then Zs = Ys
    [] X|Xr then Zr in
      Zs = X|Zr
      {Append Xr Ys Zr}
  end
end
```

Daí, vemos que nenhuma operação é realizada após a chamada recursiva, isto é, não há acúmulo de operações e o tamanho da pilha é constante, visto que $Zs = X|Zr$ pode ser colocado antes da chamada recursiva - mesmo que Zr seja uma variável não ligada - devido ao comportamento dataflow, permitindo que o algoritmo Append se comporte de maneira iterativa.

Item B)

```
fun {AppendIter Xs Ys}
  case Xs
    of nil then Ys
    [] X|Xr then {AppendIter Xr X|Ys}
  end
end
fun {append Xs Ys}
  {AppendIter {Reverse Xs} Ys} end
```

Diferentemente do append tradicional, cuja chamada recursiva se dá por $X \mid \{\text{AppendIter } Xr \text{ } Ys\}$ e não possui a otimização da chamada pela cauda, no append iterativo utilizamos $\{\text{AppendIter } Xr \text{ } X \mid Ys\}$, que concatena o inverso da lista Xs à lista Ys . O que acontece é que o algoritmo coloca o primeiro elemento de Xs à frente de Ys ($X|Ys$), depois coloca o segundo elemento de Xs à frente da lista $X|Ys$ resultante e assim por diante. Para consertar, basta trabalhar com $\{\text{Reverse } Xs\}$ no lugar de Xs ao chamar a função Append. Dessa forma, não se faz necessária a existência de variáveis dataflow, visto que a mudança de estado S já foi passada como argumento na chamada recursiva, e daí não é preciso trabalhar com valores parciais.

AP2 - 2023.2

Questão 1)

Item A)

```
proc {Append Xs Ys ?Zs}
    case Xs
        of nil then Zs = Ys
        [ ] X|Xr then Zr in
            Zs = X|Zr
            {Append Xr Ys Zr}
    end
end
```

A chamada pela cauda é possível devido ao comportamento dataflow, que possibilita o programa aguardar até que variáveis não ligadas recebam algum valor, sem suspender a execução. No caso do Append, ao expandirmos o algoritmo para a linguagem núcleo, vemos que a operação $Zs = X|Zr$ pode ser colocada antes da chamada recursiva, mesmo que Zr ainda seja uma variável não ligada. Isso permite que o Append seja um programa recursivo otimizado pela cauda, dado que não há operações após a chamada recursiva.

Item B) Resolvido na AP2 de 2024.2.

Questão 2)

Item A)

Ao tentar aplicar o delete em uma fila vazia, acontece a chamada “remoção antecipada”, isto é, o próximo elemento a ser inserido será automaticamente removido da fila, fazendo com que ela volte ao estado normal de fila vazia, com $S == E$ e $N == 0$, e continue funcionando normalmente. Isso acontece devido ao comportamento dataflow, uma vez que, ao aplicar o delete em uma lista vazia, S será uma variável não ligada, e o programa aguardará até que S receba um valor para executar a deleção. Ao inserir um elemento à fila, digamos X , como $S == E$, então S se ligará a $X|E1$ e o removerá da fila.

Item B)

Na definição do `IsEmpty`, não se pode utilizar a condição $S == E$ para verificar se uma pilha é vazia, visto que o operador de igualdade requer operandos que estejam ligados a valores para realizar a comparação, e, caso S e E sejam variáveis não ligadas, o programa será suspenso. O ideal é realizar a comparação $N == 0$, dado que N recebe valor desde a criação da lista ($N = 0$).