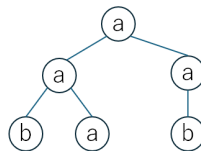


[基本プログラムコース]

練習問題 [基本プログラムコースのみ]

1. 以下のような文法で木構造が定義されているとする。ノードには a または b がラベルとして付加されている。この木 T において、a とラベル付けされたノードの個数が K であるという関係を表す述語 $n_of_nodes(T, K)$ のプログラムを作成せよ。たとえば、 $n_of_nodes(t(a, t(a, b, a), t(a, b)), K)$ は $K=4$ となって成功する (図参照)。ただし、 T にはこの定義に従う表現のみが入力されると考えてよい。

```
Tree ::= t(Node, Tree, Tree) | t(Node, Tree) | Node
Node  ::= a | b
```

**演習問題 (r11) [基本プログラムコース]**

- (1) 整数を要素とする長さ n ($n > 0$) の 2 つのリストをそれぞれ L, M とする。すべての i ($i \geq 0$) について、 L, M の i 番目の要素の和がリスト N の i 番目の要素であるという関係を表す述語 $sum_of_pair(L, M, N)$ のプログラムを作成せよ。たとえば、 $sum_of_pair([1, 2, 3], [-1, 2, 0], N)$ は $N = [0, 4, 3]$ となって成功する。
- (2) 正の整数からなるリスト L に対して、その要素で 2 桁の数の個数が N であるという関係を表す述語 $two_digits(L, N)$ のプログラムを作成せよ。
たとえば、 $two_digits([3, 14, 1, 102, 25], N)$ は $N = 2$ となって成功する。
- (3) (書籍名, 在庫の有無, 出版年, 出版社) を要素とするリスト $L1$ に対して、発行年が 2010 年以降で在庫がある書籍のみを集めてその出版社とともに (書籍名, 出版社) という組をつくって要素としたリストが $L2$ であるという関係を表す述語 $book_list(L1, L2)$ のプログラムを作成せよ。たとえば、
 $book_list([(b1, no, 2017, pb1), (b2, yes, 1998, pb2), (b3, yes, 2015, pb3), (b4, no, 2000, pb4)], L2)$ は $L2 = [(b3, pb3)]$ となって成功する。
- (4) 以下の文法 (G) が定義されているとき、与えられた表現 X が式 (Expr) であることを判定する $isExp(X)$ のプログラムを作成せよ。たとえば $isExp(plus(0, suc(0)))$ は成功し、 $isExp(suc(plus(0, 0)))$ は失敗する。

 (G)

```
Expr ::= plus(Expr, Term) | minus(Expr, Term) | Term
Term ::= 0 | suc(Term)
```

- (5) 以下のような文法で木構造が定義されているとする．このように定義された木 T1 が与えられたとき，ノードのラベル a をすべて c に変更して得られるものが木 T2 であるという関係を表す `subst_label( T1, T2 )` のプログラムを作成せよ．
 たとえば，`subst_label( t( a, t(b,a,b), t(a,b) ), T2 )` は `T2 = t( c, t(b c, b), t(c, b) )` となって成功する (図 11.1 参照)．ただし，T1 には以下の木構造に従う表現のみが入力されると考えてよい．

```
Tree ::= t(Node,Tree,Tree) | t(Node,Tree) | Node
Node  ::= a | b
```

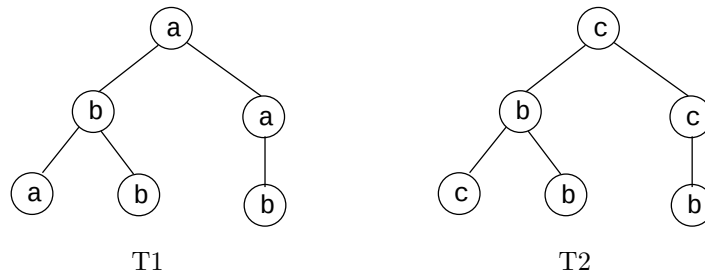


図 11.1

リストの入れ子

■ リストの平坦化 (末端再帰による実装)

第1リストが長いと効率が悪い (append の多用)

```
flatten( [], [] ).
flatten( [X|Xs], Ys ) :-
    flatten( X, X1 ), flatten( Xs, X2 ), append( X1, X2, Ys ).
flatten( X, [X] ).      % X がリストでない場合 (アトムなど)
```

```
append( [], Y, Y ).
append( [X|Xs], Y, [X|Zs] ) :- append( Xs, Y, Zs ).
```

例: `flatten( [a,[b,[c]],[],d], [a,b,c,d] )` は成功する.

差分リスト (d-list)

- リストを2つのリストの差分で表現する方法.
- リストの接続において、完全に具体化されたリスト同士をつなぐのではなく、尾部がのりしろになった (変数のままになっている) リストをつなぐ.
- 通常のリスト表現による接続では、少しずつ完全なリストをつくってから `append` でつなぐため、何度もリストを走査する必要があり非効率的.
- 不完全なリスト同士を頭部と尾部を一致させるだけでつなぐため、何度もリストを走査する必要がなく、特に接続するリスト数が増えると効果は大きい.
- text p.201 8.5.3 節参照.

リスト `[a,b,c]` の差分リストによる表現 (ただし `T` は任意のリスト)

```
[a,b,c]-[]
[a,b,c,d,e]-[d,e]
[a,b,c | T]-T      など
```

リスト `[]` の差分リストによる表現 (ただし `T` は任意のリスト)

```
T-T
```

■ リストの平坦化 (差分リストによる実装)

```
flatten2( X, Y ) :- flatten_dl( X, Y-[] ). % まず差分リストの構造をつくる
```

```
flatten_dl( [], T-T ).      % 入力リストを最後まで見たら、出力リストも閉じる
flatten_dl( [X|Xs], H-T ) :-
    flatten_dl( X, H-T1 ), flatten_dl( Xs, T1-T ). % T1 がのりしろになる
flatten_dl( X, [X|T]-T ). % 定数の場合はそのまま出力リストにいれる
```

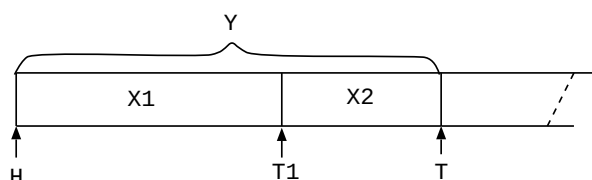


図 5.1

演習問題 (r11) [応用プログラムコース]

基本的に差分リストと関係なく解ける復習問題である。* のついている問題はオプション課題。

- (6) 入れ子を含むリスト L に与えられた項 A が出現するかどうかを判定する述語 `ocr_check_list( A, L )` のプログラムを作成せよ。flatten を使わないこと。たとえば, `ocr_check_list( a, [b,[c,a]] )` は成功し, `ocr_check_list( a, [b,[c]] )` は失敗する。
- (7) アトムを要素として入れ子を含むリスト L に与えられた項 A が出現する回数が C であるという関係を表す述語 `count_ocr_list( A, L, C )` のプログラムを作成せよ。組み込み述語である `atom` を使用してよい。flatten を使わないこと。たとえば, `count_ocr_list( a, [a,b,[c,a],a], C )` は $C = 3$ となって成功する。
- (8) 入れ子を含まないリスト $L1$ に出現するアトム A をすべてアトムのリスト Bs の中身に書き換えた結果がリスト $L2$ であるような関係を表す述語 `rename_atom_to_list( L1, A, Bs, L2 )` のプログラムを作成せよ。他の述語を使わず, 再帰的に定義すること。flatten を使わないこと。たとえば, `rename_atom_to_list( [a,b,c,a], a, [z,zz], L2 )` は $L2 = [z,zz,b,c,z,zz]$ となって成功する。
- (9) green, white, red の3種類 (それぞれ g,w,r と略記する) の複数の要素から成るリスト $L1$ を並べかえて, 最初に green, 次に white, 最後に red が現われるようなリストが $L2$ であることを表す述語 `flag( L1, L2 )` の再帰型プログラムを作成せよ。ただし, 各要素は“色 (順番)”という形で表現し, 同一色同士においては出現の順序は保持されるものとする。flatten を使わないこと。たとえば, `flag( [g(1),w(2),r(3),g(4),w(5)], L2 )` は $L2 = [g(1),g(4),w(2),w(5),r(3)]$ となって成功する。
- (10) 演習問題 r10(9) のプログラムを改訂して閉路のあるグラフに対しても有効であるような縦型探索を行う `dfs_loop(Path)` の繰り返し型プログラムを作成せよ。さらに, 得られたすべての経路の集合が `PathList` であるという関係を表す述語 `dfs_loop_list(PathList)` のプログラムを `setof` を使って作成せよ。図 11.2 のグラフで全解 (各ノードを高々1回しか通らないもののみを解とする) が求まることを確認せよ。たとえば, `dfs_loop(Path)` の解の1つは $Path=[g, f, d, c, a, s]$ であり, これを含めて合計7個の解がある。なお, 解の表示が省略される場合は, `test:- dfs_loop(Path),write(Path).` として `test` を実行するとよい。

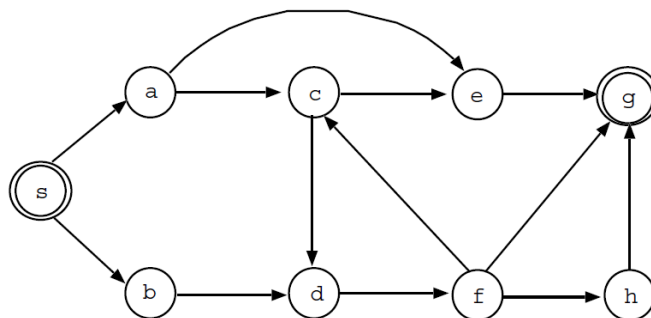


図 11.2

- (11)* (9) の `rename_atom_to_list( L1, A, Bs, L2 )` のプログラムの差分リスト版を作成せよ。トップレベルを `rename_atom_to_list_dl( L1, A, Bs, L2 ) :- rename2( L1, A, Bs, L2-[] )` . として, `rename2` を定義せよ。

(12)* (10) の 差分リスト版を作成せよ. トップレベルを `flag_dl( Xs, G ) :- fl( Xs, G-W, W-R, R-[] )`. として, `fl` を定義せよ.