Задачи для написания спецификаций на языке RSL

Для каждой представленной ниже функции написать спецификацию (явную или неявную) на языке RSL **без использования рекурсии, циклов и аксиом**. Старайтесь искать как можно более короткую запись спецификации.
Нерешенные на семинаре задачи переходят в домашнюю работу.

1. «сортировка»: отсортировать данный список чисел
   ```
   sort: Real-list -> Real-list
   ```

2. «все подстроки»: для данной строки выдать множество всех ее подстрок
   ```
   allsubstrs: Text -> Text-set
   ```

3. «задача о рюкзаке»: дано множество предметов, информация о их весе, ограничение на вес рюкзака; построить распределение данного множества предметов по рюкзакам (множество множеств предметов) так, чтобы минимизировать количество рюкзаков, учитывая максимальный допустимый вес рюкзака
   ```
   rucksack: (T-set) >< (T-m->Nat) >< Nat -~-> T-set-set
   ```

4. «объединение классов»: дано множество множеств объектов (например, множество классов эквивалентности объектов), по данным двум объектам, принадлежащим разным классам, построить новое множество классов эквивалентности, отличающееся от данного слиянием классов, которым соответствуют данные объекты
   ```
   type A, P = (A-set)-set
   value mergeC: P >< A >< A -~-> P
   ```

5. «разметка текста-I»: в данном тексте заменить все вхождения данного символа на другой символ.
   ```
   replace1: Text >< Char >< Char -> Text
   ```

6. «разметка текста-II»: в данном тексте заменить все вхождения данной строки на другую строку.
   ```
   replace2: Text >< Text >< Text -> Text
   ```

7. «слияние»: объединить элементы двух отсортированных списков для получения отсортированного списка
   ```
   merge: Real-list >< Real-list -~-> Real-list
   ```

8. «выделение слов»: для данного текста и символа-разделителя вернуть множество всех слов (словом считается подстрока текста, разделенная от других слов символом-разделителем)
   ```
   parse: Text >< Char -> Text-set
   ```

9. «факториал без рекурсии»: для данного натурального числа вернуть его факториал.
```
f: Nat -> Nat
```

10. «инъектирование»: вставить в данный текст после каждого вхождения некоторого символа другой символ
```
inject: Text >< Char >< Char -> Text
```

11. «представители»: для каждого символа данной строки оставить первое его вхождение для получения результирующей строки
```
m: Text -> Text
```

В задачах 12-18 предлагается использовать следующее представление графа:
```
type V, G = V -m-> V-set
```
(отображение из вершины в множество инцидентных ей вершин). Правда, такое определение типа надо еще снабдить ограничением — подумайте, каким.

12. «поиск пути в графе»: для данного графа и двух вершин вернуть путь из первой во вторую.
```
path : G >< V >< V -~-> V-list
```

13. «гамильтонов путь»: для данного графа вернуть гамильтонов путь в нем.
```
gpath : G-~-> V-list
```

14. «деревянность»: проверить, является ли данный граф деревом.
```
istree: G -> Bool
```

15. «ограниченная достижимость-I”: построить множество вершин данного графа, достижимых из данной вершины с не более 1 промежуточной вершиной.
```
med1: G >< V -~-> V-set
```

16. «ограниченная достижимость-N»: построить множество вершин данного графа, достижимых из данной вершины с не более *N* промежуточной вершиной.
```
medN: G >< V >< Nat -~-> V-set
```

17. «вставка в дерево»: в данный граф-дерево поиска данный элемент для получения корректного дерева поиска
```
type V = Nat
value insert: G >< V -~-> G
```

18. «инвертирование дуг»: для данного графа построить граф из тех же вершин, в котором каждой дуге из вершины X в вершину Y взаимнооднозначно сопоставлена дуга данного графа из вершины Y в вершину X.
```
invert: G -> G
```