Heinrich Heine Universität Düsseldorf

Software Engineering and Programming Languages
Prof. Dr. Michael Leuschel
David Geleßus, Fabian Vu

Introduction to Logic Programming – WS 2023
Exercise Sheet 12

1 Exercises

Exercise 1 (Lecture – Inverted Classroom)
Watch the lecture videos *21 CLPFD*[1] in the HHU Mediathek. The corresponding slides are uploaded in ILIAS: 13_CLPFD.pdf

The complete playlist is available at: `https://mediathek.hhu.de/playlist/691`.

Note: you have to log in with your HHU account (Uni-Kennung) to see the lecture videos!

The exercises will be discussed on 23rd January 2024.

Exercise 2 (Difference Lists)
As you have seen in the lecture, difference lists can improve performance of list operations. In particular, we are able to add an element to the end of a difference list in constant time while usual Prolog lists show linear time complexity for this operation. In general, you can choose an arbitrary structure to store a difference list. A common representation uses the functor `-/2` such as `[a,b|R]-R`.

a) Implement a predicate `toDL(+List, ?DiffList, ?Diff)` which transforms a Prolog list into a difference list. The first argument is the Prolog list and the second and third arguments represent the difference list.

Examples:

```
?- toDL([a,b,c], X, R).
X = [a,b,c|R]
?- toDL([a,b],[a,b,c],[c]).
true
?- toDL([a,b],X, [c,d]).
X = [a,b,c,d]
```

b) Implement a predicate `dlconcat(+DL1, +DL2, -DLOut)` which concatenates two difference lists represented as `X-Y`.

[1] `https://mediathek.hhu.de/watch/9d7f37e0-b068-4cf8-a75a-4ca0c5de89d0`

Examples:

```
?- dlconcat([a,b,c|A]-A,[d,e,f|B]-B, List).
A = [d,e,f|B],
List = [a,b,c,d,e,f|B]-B
```

c) Implement a predicate `dlmember/2` which behaves as `lists:member/2` but for difference lists.

Exercise 3 (Definite Clause Grammars (DCGs))
Implement a (dcg-)parser which is able to detect if a term is correctly bracketed. If this is the case, the amount of bracket pairs should be counted.

Examples:

```
?- parse_pars(‘{([<>])}‘, N).
N = 4
?- parse_pars(‘(‘, N).
false
```

Exercise 4 (Parser - Sudoku)
In this exercise we want to implement a (dcg-)parser which is able to parse Sudoku puzzles from files.

The input is represented in the „SuDoKu Solver Format“. Blocks are bounded to the left and right by „| “. The bottom of a block is bounded by „-“ except of in the last line of blocks. Crosspoints are indicated by „+“. The numbers are split by blankspaces.

The entries of the Sudoku puzzles are numbers from the interval $[1, n]$, where n is the amount of columns and rows. A variable is represented by `X` (we introduce an anonymous variable for each `X`).

Example for $n = 9$:

```
. . . | . . . | . . .
. . . | . . . | . . .
. . . | . . . | . . .
------+-------+------
. . . | . . . | . . .
. . . | . . . | . . .
. . . | . . . | . . .
------+-------+------
. . . | . . . | . . .
. . . | . . . | . . .
. . . | . . . | . . .
```

You can find 3 test files in the folder `sudoku_puzzles/`.

We are able to read input from a file in Prolog by using `phrase_from_file/2`.

The call `phrase_from_file(sudoku_input(Sudoku), FilePath)` reads the file from `FilePath` using the definite clause grammar `sudoku_input/1`. The result is unified with the variable `Sudoku`.

The task is to implement the definite clause grammar `sudoku_input/1` for which we provide an entry point in the corresponding test file.

Hint: The separating rows between blocks should be skipped when parsing a puzzle. This results in empty lists which have to be removed from the output returned by the DCG. You might want to implement a predicate `remove_empty_lists/2` which is able to do so.