Middle East Technical University   Department of Computer Engineering

CENG 242

Programming Language Concepts

Spring '2017-2018

Programming Assignment 5

Due date: 27 May 2018, Saturday, 23:55

1 Introduction

In this assignment, you are going to use Prolog in order to solve a basic hardware configuration task where you put hardware components into the sections of a computer box. You will be given the structure of the box and some constraints regarding the components. Your task is to find a configuration of these hardware components so that the given constraints are satisfied.

2 Hardware Configuration Task

This configuration task consists of assignments of the given computer hardware components to the given grid-like computer box. The computer box contains square shaped sections that are labeled. Two sections are adjacent if they share an edge. An example computer box sketch is given in Figure 1.

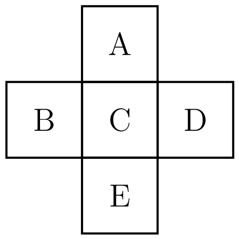

Figure 1: Example computer box with a plus-like shape. Each section is labeled with letters from A to E.

The task requires you to assign the given set of hardware components, like CPU, fan, RAM, HDD etc., to the sections of the given computer box. There will be two type of constraints regarding these components; (1) two components may be required to be *close to* each other and (2) a component may be required to have an *outer edge*. Your solution should find all possible configurations given these constraints.

3 Specifications

3.1 Hardware Representations

The list of sections and the structure of the computer box will be given in a file named `hardware.pl`.

3.1.1 `sections/1` predicate

- The list that contains the section labels is given as;

```
sections(SECTION_LIST).
```

  where `SECTION_LIST` represents the list of the section labels given as atoms.

- The fact can be read as "`SECTION_LIST` *is the list of the sections* in the computer box".
- There will be no duplicates in `SECTION_LIST`.
- There will be only one `sections` fact in the given `hardware.pl` file.

3.1.2 `adjacent/2` predicate

- The adjacency between two sections is given;

```
adjacent(SECTION1, SECTION2).
```

  where `SECTION1` and `SECTION2` represent the labels of the sections that are adjacent, i.e., sharing an edge.

- The fact can be read as "`SECTION1` *is adjacent to* `SECTION2`".
- `SECTION1` and `SECTION2` are members of `SECTION_LIST`.
- In the given adjacency facts, `SECTION1` will NOT be equal to `SECTION2`.
- This predicate represents a two way connection, that is, by the definition, `SECTION1` is adjacent to `SECTION2` if and only if `SECTION2` is adjacent to `SECTION1`. In the given `hardware.pl`, only one adjacency fact will be given for a pair of sections.
- A section can be adjacent to at least one and at most four sections. That is, there is no disconnected section of the computer box, and a section can share at most four edges (due to the square shape).

3.1.3 `outer_edge/1` predicate

- The constraint on a hardware component regarding having an outer edge is given as;

```
outer_edge(COMPONENT).
```

  where `COMPONENT` represents the hardware component that should be placed on a section with at least one outer edge, i.e. an edge which is not shared by any other section.

- The fact can be read as "`COMPONENT` *should have an outer edge* in the configuration".
- `COMPONENT` is a member of the given `COMPONENT_LIST`.

3.1.4 `close_to/2` predicate

- The constraint on two hardware components regarding being close to each other is given as;

```
close_to(COMPONENT1, COMPONENT2).
```

  where `COMPONENT1` and `COMPONENT2` represent the two components that must be placed close to each other, that is, their assigned sections must share an edge.

- The fact can be read as "COMPONENT1 *should be placed close to* COMPONENT2 in the configuration".
- COMPONENT1 and COMPONENT2 are members of the given COMPONENT_LIST.
- In the given closeness facts, COMPONENT1 will NOT be equal to COMPONENT2.
- This predicate represents a two way connection, that is, if COMPONENT1 should be close to COMPONENT2, then, by the definition, COMPONENT2 should also be close to COMPONENT1. In the given list of constraints, only one closeness fact will be given for a pair of components.

3.1.5 put/2 predicate

- A placement of a hardware component to a computer box section is represented as;

```
put(COMPONENT, SECTION).
```

  where COMPONENT and SECTION are the hardware component that is placed and the section place of this component in the computer box.
- The fact can be read as "COMPONENT *is put to* SECTION in the configuration".
- COMPONENT is a member of the given COMPONENT_LIST.
- SECTION is a member of SECTION_LIST.

3.2 Task

You are going to write a predicate, **configuration/3**, that will find a placement for the given set of hardware components onto the given computer box under the given constraints.

3.2.1 configuration/3 predicate

- The predicate will take 3 arguments.

```
configuration(COMPONENT_LIST, CONSTRAINT_LIST, PLACEMENT_LIST).
```

- COMPONENT_LIST contains the hardware components that are going to be placed into the sections.
- CONSTRAINT_LIST is made of two types of constraints; outer_edge/1 and close_to/2 predicates, defined over the given components.
- PLACEMENT_LIST consists of the placements of each component into the sections of the computer box. The placements are represented with put/2 predicates.
- COMPONENT_LIST will NOT have any duplicates, i.e. no component can occur in the list more than once.
- It is guaranteed that there will be at most one constraint regarding a component in the given CONSTRAINT_LIST. That is, a component can only be seen in at most one constraint in the list.
- It is guaranteed that COMPONENT_LIST and CONSTRAINT_LIST arguments will be provided in any kind of query.
- A query missing PLACEMENT_LIST should find a configuration of the given components under the given constraints and unify PLACEMENT_LIST.
- In the final PLACEMENT_LIST, all the components must be placed to a section.

- A query providing all of the arguments should evaluate as `true` if the given `PLACEMENT_LIST` is a valid configuration of the given `COMPONENT_LIST` under the given `CONSTRAINT_LIST`.
- If there is no valid configuration under the constraints, the predicate should evaluate as `false`.
- If the given `COMPONENT_LIST` is empty, `PLACEMENT_LIST` should also be empty.
- When forced to backtrack, the predicate should be able to find all valid configurations.
- The order in the `PLACEMENT_LIST` is NOT important.

3.3 An Example

3.3.1 Example Knowledge Base

An example `hardware.pl` file for the computer box given in Figure 1 is represented in Prolog as;

```
:- module(hardware, [sections/1, adjacent/2]).

sections([sA, sB, sC, sD, sE]).

adjacent(sA,sC).
adjacent(sB,sC).
adjacent(sC,sD).
adjacent(sC,sE).
```

where the sections are given by atoms as `sA`, `sB`, `sC`, `sD`, `sE`.

3.3.2 Example Runs for `configuration/3`

```
?- configuration([], [], PlacementList).
PlacementList = [].

?- configuration([cpu], [], PlacementList).
PlacementList = [put(cpu, sA)] ;
PlacementList = [put(cpu, sB)] ;
PlacementList = [put(cpu, sC)] ;
PlacementList = [put(cpu, sD)] ;
PlacementList = [put(cpu, sE)].

?- configuration([fan], [outer_edge(fan)], PlacementList).
PlacementList = [put(fan, sA)] ;
PlacementList = [put(fan, sB)] ;
PlacementList = [put(fan, sD)] ;
PlacementList = [put(fan, sE)].

?- configuration([cpu, ram], [close_to(cpu, ram)], PlacementList).
PlacementList = [put(cpu, sA), put(ram, sC)] ;
PlacementList = [put(cpu, sB), put(ram, sC)] ;
PlacementList = [put(cpu, sC), put(ram, sB)] ;
PlacementList = [put(cpu, sC), put(ram, sA)] ;
PlacementList = [put(cpu, sC), put(ram, sE)] ;
PlacementList = [put(cpu, sC), put(ram, sD)] ;
PlacementList = [put(cpu, sD), put(ram, sC)] ;
PlacementList = [put(cpu, sE), put(ram, sC)] ;
false.

?- configuration([cpu, ram, ssd, hdd, fan, gpu], [], PlacementList).
false.

?- configuration([cpu, ram, ssd, hdd, fan], [outer_edge(fan), close_to(cpu, ram), close_to(ssd, hdd)], ↩
     PlacementList).
false.

?- configuration([cpu, ram, ssd, hdd, fan], [outer_edge(fan), close_to(cpu, ram)], PlacementList).
PlacementList = [put(cpu, sA), put(ram, sC), put(ssd, sB), put(hdd, sD), put(fan, sE)] ;
PlacementList = [put(cpu, sA), put(ram, sC), put(ssd, sB), put(hdd, sE), put(fan, sD)] ;
PlacementList = [put(cpu, sA), put(ram, sC), put(ssd, sD), put(hdd, sB), put(fan, sE)] ;
PlacementList = [put(cpu, sA), put(ram, sC), put(ssd, sD), put(hdd, sE), put(fan, sB)] ;
PlacementList = [put(cpu, sA), put(ram, sC), put(ssd, sE), put(hdd, sB), put(fan, sD)] ;
PlacementList = [put(cpu, sA), put(ram, sC), put(ssd, sE), put(hdd, sD), put(fan, sB)] ;
PlacementList = [put(cpu, sB), put(ram, sC), put(ssd, sA), put(hdd, sD), put(fan, sE)] ;
PlacementList = [put(cpu, sB), put(ram, sC), put(ssd, sA), put(hdd, sE), put(fan, sD)] ;
PlacementList = [put(cpu, sB), put(ram, sC), put(ssd, sD), put(hdd, sA), put(fan, sE)] ;
PlacementList = [put(cpu, sB), put(ram, sC), put(ssd, sD), put(hdd, sE), put(fan, sA)] ;
PlacementList = [put(cpu, sB), put(ram, sC), put(ssd, sE), put(hdd, sA), put(fan, sD)] ;
PlacementList = [put(cpu, sB), put(ram, sC), put(ssd, sE), put(hdd, sD), put(fan, sA)] ;
PlacementList = [put(cpu, sC), put(ram, sB), put(ssd, sA), put(hdd, sD), put(fan, sE)] ;
PlacementList = [put(cpu, sC), put(ram, sB), put(ssd, sA), put(hdd, sE), put(fan, sD)] ;
PlacementList = [put(cpu, sC), put(ram, sB), put(ssd, sD), put(hdd, sA), put(fan, sE)] ;
PlacementList = [put(cpu, sC), put(ram, sB), put(ssd, sD), put(hdd, sE), put(fan, sA)] ;
PlacementList = [put(cpu, sC), put(ram, sB), put(ssd, sE), put(hdd, sA), put(fan, sD)] ;
PlacementList = [put(cpu, sC), put(ram, sB), put(ssd, sE), put(hdd, sD), put(fan, sA)] ;
PlacementList = [put(cpu, sC), put(ram, sA), put(ssd, sB), put(hdd, sD), put(fan, sE)] ;
PlacementList = [put(cpu, sC), put(ram, sA), put(ssd, sB), put(hdd, sE), put(fan, sD)] ;
PlacementList = [put(cpu, sC), put(ram, sA), put(ssd, sD), put(hdd, sB), put(fan, sE)] ;
PlacementList = [put(cpu, sC), put(ram, sA), put(ssd, sD), put(hdd, sE), put(fan, sB)] ;
PlacementList = [put(cpu, sC), put(ram, sA), put(ssd, sE), put(hdd, sB), put(fan, sD)] ;
PlacementList = [put(cpu, sC), put(ram, sA), put(ssd, sE), put(hdd, sD), put(fan, sB)] ;
PlacementList = [put(cpu, sC), put(ram, sE), put(ssd, sA), put(hdd, sB), put(fan, sD)] ;
PlacementList = [put(cpu, sC), put(ram, sE), put(ssd, sA), put(hdd, sD), put(fan, sB)] ;
PlacementList = [put(cpu, sC), put(ram, sE), put(ssd, sB), put(hdd, sA), put(fan, sD)] ;
PlacementList = [put(cpu, sC), put(ram, sE), put(ssd, sB), put(hdd, sD), put(fan, sA)] ;
PlacementList = [put(cpu, sC), put(ram, sE), put(ssd, sD), put(hdd, sA), put(fan, sB)] ;
PlacementList = [put(cpu, sC), put(ram, sE), put(ssd, sD), put(hdd, sB), put(fan, sA)] ;
PlacementList = [put(cpu, sC), put(ram, sD), put(ssd, sA), put(hdd, sB), put(fan, sE)] ;
PlacementList = [put(cpu, sC), put(ram, sD), put(ssd, sA), put(hdd, sE), put(fan, sB)] ;
PlacementList = [put(cpu, sC), put(ram, sD), put(ssd, sB), put(hdd, sA), put(fan, sE)] ;
PlacementList = [put(cpu, sC), put(ram, sD), put(ssd, sB), put(hdd, sE), put(fan, sA)] ;
PlacementList = [put(cpu, sC), put(ram, sD), put(ssd, sE), put(hdd, sA), put(fan, sB)] ;
PlacementList = [put(cpu, sC), put(ram, sD), put(ssd, sE), put(hdd, sB), put(fan, sA)] ;
PlacementList = [put(cpu, sD), put(ram, sC), put(ssd, sA), put(hdd, sB), put(fan, sE)] ;
PlacementList = [put(cpu, sD), put(ram, sC), put(ssd, sA), put(hdd, sE), put(fan, sB)] ;
PlacementList = [put(cpu, sD), put(ram, sC), put(ssd, sB), put(hdd, sA), put(fan, sE)] ;
PlacementList = [put(cpu, sD), put(ram, sC), put(ssd, sB), put(hdd, sE), put(fan, sA)] ;
PlacementList = [put(cpu, sD), put(ram, sC), put(ssd, sE), put(hdd, sA), put(fan, sB)] ;
PlacementList = [put(cpu, sD), put(ram, sC), put(ssd, sE), put(hdd, sB), put(fan, sA)] ;
PlacementList = [put(cpu, sE), put(ram, sC), put(ssd, sA), put(hdd, sB), put(fan, sD)] ;
PlacementList = [put(cpu, sE), put(ram, sC), put(ssd, sA), put(hdd, sD), put(fan, sB)] ;
PlacementList = [put(cpu, sE), put(ram, sC), put(ssd, sB), put(hdd, sA), put(fan, sD)] ;
PlacementList = [put(cpu, sE), put(ram, sC), put(ssd, sB), put(hdd, sD), put(fan, sA)] ;
PlacementList = [put(cpu, sE), put(ram, sC), put(ssd, sD), put(hdd, sA), put(fan, sB)] ;
PlacementList = [put(cpu, sE), put(ram, sC), put(ssd, sD), put(hdd, sB), put(fan, sA)] ;
false.

?- configuration([cpu, ram, ssd, hdd, fan], [outer_edge(fan), close_to(cpu, ram)], [put(cpu, sD), put(ram, sC),↩
      put(ssd, sB), put(hdd, sE), put(fan, sA)]).
true.
```

4 Regulations

1. **Programming Language:** You should write your code using SWI Prolog.
2. **Late Submission:** See the syllabus for the details.
3. **Cheating:** All the work should be done individually. **We have zero tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations.
4. **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.
5. **Evaluation:** Your program will be evaluated automatically using "black-box" technique so make sure to obey the specifications.

5 Submission

Submission will be done via Ceng Class. Submit a single file called "hw5.pl" through the Ceng Class system. The file MUST start with the following lines.

```
:- module(hw5, [configuration/3]).
:- [hardware].
```