Assignment 1: Developing a Language for Ordering Computing Hardware from a Vendor

2IMP20 - DSL Design

May 12, 2022

1 Introduction

The goal of this assignment is to design and implement—using a language workbench—a Domain-Specific Language (DSL) for ordering computing hardware from a vendor. This assignment must be implemented using the Rascal language workbench. Make sure to first read and follow the separate document with instructions on how to install the Rascal language workbench.

Then,

- Clone[1] the assignment's project from the repository.
- Import the assignment's project into the current workspace (File ⟶ Import ⟶ Existing Projects into Workspace). Then, select the location where the project's repository was cloned.

2 Assignment

The first assignment is meant to get you acquainted with the basics of defining languages using Rascal. The goal of the assignment is to build a DSL called HCL (Hardware Configuration Language) for defining computing hardware resources (to order from a vendor). One of the tasks in this assignment is to develop a concrete syntax (using Rascal's grammar formalism for describing languages) for HCL, which can be used to parse HCL programs.

In the repository [2] you will find an Eclipse project with the skeleton of the assignment.

NB: When opening this project, it may get stuck on the Rascal builder, which can not execute because of necessary Maven updates which in turn may be blocked by the Rascal builder. In that case, stop the Rascal builder in the progress bar of Eclipse, and restarting Eclipse now seems to fix the problem.

Take a look at the `src` folder within Eclipse. This folder contains the language modules and each language module contains the instructions for the exercises.

[1]Documentation on how to clone a repository is available here.

[2]`https://github.com/Daanchaam/DSL-Design-2022-Assignment-1`

HCL allows users to define hardware resources by defining instances, and each instance has different properties. We will use HCL to demonstrate the various aspects of language design using a language workbench (Rascal). Hereafter, we introduce more details about the HCL language.

2.1 Grammar

A computer resource may have a label and contains a list of components. Each component provides information describing the hardware involved, and it can be of one of three types: *processing* or *storage* or *display*.

A *processing* component has a number of cores, a clock speed (in GHz), and L1, L2, L3 CPU caches (in KiB or MiB). Cores and memory are modelled using positive integer values; clock speed using a positive real value.

A *storage* component is defined by its type and size. There are two types of storage: Hard Disk Drive (HDD) or Solid State Drive (SSD). The storage's size is defined by an integer value that ranges from 32 until 1024 GB.

A *display* component is defined by its diagonal size (in inches) and an indicator of its number of pixels (one of HD, Full-HD, 4K, 5K).

One important feature of HCL is that it allows users to reuse existing components. This can be achieved simply by using the component's label.

2.2 Well-formedness Resources

In order to have a valid HCL hardware definition, some requirements have to be satisfied. These requirements can not be expressed in context-free grammars. The following conditions ensure the well-formedness of a HCL hardware definition.

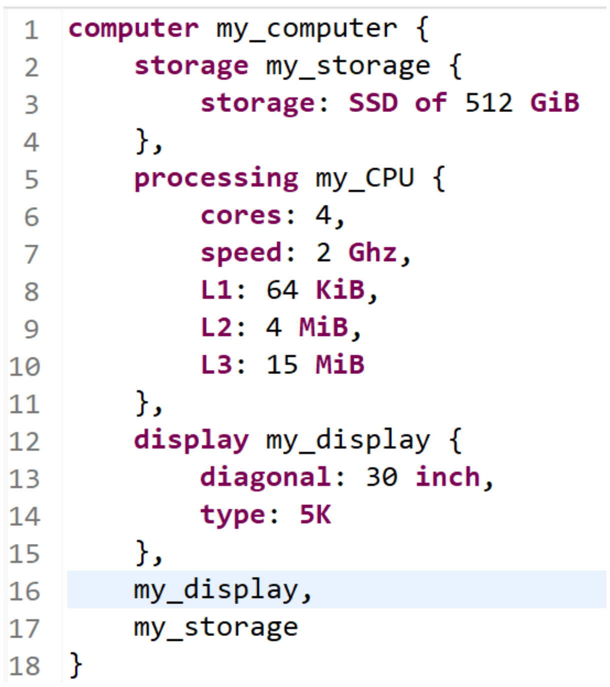
- Components' labels must be unique.
- The total storage size must be greater than zero but less than or equal to 8192 GiB.
- The maximum L1 size is 128 KiB, the maximum L2 size is 8 MiB, the maximum L3 size is 32 MiB; and their sizes must satisfy L1 < L2 < L3.
- Display type must be valid. In other words, the type has to be one of the four mentioned in Section 2.1.
- Do not accept duplicate components with the exact same configuration and different labels.
- The language supports Booleans, integers, reals, and string types.

Figure 1 shows a valid simple hardware definition using HCL language.

3 Deliverable

The first assignment consists of four parts:

- Define a concrete syntax of HCL using Rascal's grammar formalism (module `Syntax.rsc`).



```
computer my_computer {
    storage my_storage {
        storage: SSD of 512 GiB
    },
    processing my_CPU {
        cores: 4,
        speed: 2 Ghz,
        L1: 64 KiB,
        L2: 4 MiB,
        L3: 15 MiB
    },
    display my_display {
        diagonal: 30 inch,
        type: 5K
    },
    my_display,
    my_storage
}
```

Figure 1: Example of a HCL program.

- Define a parse function for HCL. The name of the function is `parserHCL(...)`. It gets a location (`loc`) as parameter and it returns a concrete HCL resource (module Parser.rsc).
- Define an abstract syntax for HCL (module `AST.rsc`).
- Define the function `cst2ast(...)`, which takes a parse tree of a HCL resource as parameter and returns an abstract syntax tree as described in the AST (module `CST2AST.rsc`).
- Specify a well-formedness checker for HCL.
  Define the function `checkHardwareConfiguration(...)`, which takes the AST of a resource as parameter and verifies that all well-formedness checks pass (module `Check.rsc`).

4 Testing

The assignment's skeleton contains a module called `Plugin.rsc`. This module registers the HCL into Eclipse. This means that Eclipse will recognize files with extension `.hcl`, and it will call the HCL's parser. If the program is syntactically correct, you should observe syntax highlighting on it. If that is not the case, it is highly probable that the program contains a parse error. To activate such functionality you have to open Rascal's REPL, import the module `hcl::Plugin`, and call the `main()` function. Likewise, this module also contains a function called `checkWellformedness()` that receives as a parameter the path of a HCL program and returns a Boolean value. If the program is correct, it returns true or false if the program is not well-formed.

Now, you can create your first `myfirst.hcl` file. Open it and enter your first hardware specification using HCL. Observe what happens if you write a syntactical correct and incorrect HCL specifications.

5 Submission

You have to submit a zip file containing:

- Your HCL language solution including all modified files.
- The test programs demonstrating the correct parsing of non-trivial HCL specifications.
- Test programs containing well-formedness check errors. It is also important to add comments to the files explaining the modifications/extensions you have made. You have to submit this zip file *as a Canvas group of two students* before 23:59 of Monday 30th of May via Canvas.