Department of Computer Science 3
RWTH Aachen University

Software Language Engineering, Summer 2020
Exercise-Sheet 1

Exercise Sheet 1: Developing Grammars

Organizational issues

Keep in mind that the chosen **representative** of your group hands in your solution as Moodle does **not yet support group solutions**. Always make sure to include your names in your solution.
Hint:
If you have questions/problems solving this task, use the reference manual available here:
http://www.se-rwth.de/publications/MontiCore-5-Language-Workbench-Edition-2017.pdf
to find further explanations.

Exercise 1.1: Develop your own grammar (6 Points)

In this task you will develop your own grammar "`Appointments`". Unzip the provided project and make yourself familiar with the given language project. Similar to the getting started Automaton language the grammar file to start with is located in "`src/main/grammars`". Furthermore, there are 4 example models in "`src/main/resources/examples`" that are valid models of the grammar you are asked to develop in this task.

a) Extend the given grammar such that the given example appointments can be parsed. Keep in mind that similar appointment models, e.g. different time slots, other participants etc. must be supported as well. For this only use the Nonterminals provided, i.e. do not create new nonterminal productions but develop the bodies of the given nonterminal productions (und thus their composition).
b) Extend the JUnit Test `AppointmentsParserTest` such that it comprises one testcase per example ensuring these examples are models of your developed language. For each model instantiate the parser, parse one of the models and check whether the resulting Optional is present.

Exercise 1.2: Improve your grammar (4 Points)

Now that you have a working grammar it's time to flexibilize it!

a) Introduce flexibility in modelling appointments by using an interface nonterminal for the appointment's elements such as break, participants and repetition. For this purpose, create a new grammar `AppointmentsFlexibilized`. Adapt the existing nonterminal productions to use/implement the interface nonterminal. Create a JUnit test `AppointmentsFlexibilizedParserTest` that test the new parser for the models in "`src/main/resources/example`" as well as the model in "`src/main/resources/flexibilized`".

Exercise 1.3: Extend your language (6 Points)

a) Create a new grammar and name it `Calendars`. A model of this grammar holds an owner and a list of appointments. Create an extension point to delay the decision on how concrete appointments are modelled. A model without any Appointment could be modelled as follows:

```
Peter`s calendar:
```

where “Peter” is the customizable owner.

b) Create a third grammar `CalendarsWithAppointments` that fills the extension point in your `Calendars` grammar with the Appointment Nonterminal of the Grammar `Appointments`. Create a JUnit Test `CalendarsParserTest` with one parser test for each of the following models located in “src/main/resources/calendar” that should be valid models of your language:

```
Peter`s calendar:

appointment "Kuchen" {
  start: 01/10/19 13:00
  end:    13:30
  participants: "Peter"
  once
}
```

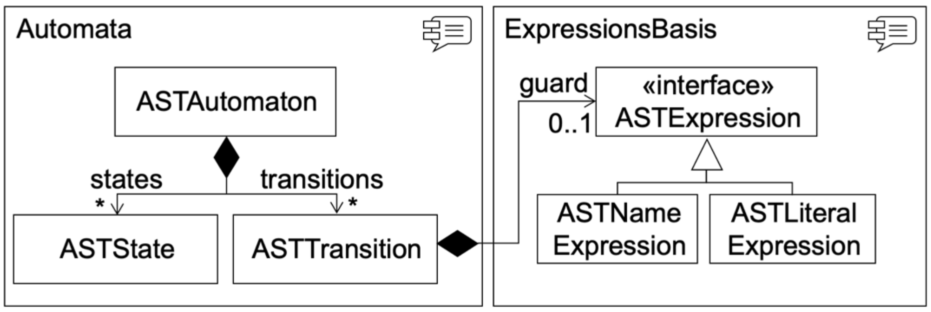
```
Tina`s calendar:

appointment "Konferenz" {
  start : 01/10/19 8:15
  end   : 02/10/19 16:15
  participants: "Peter Stein", "Paula Schmidt", "Tina Berg (Personal)"
  weekly
}

appointment "Workshop" {
  start : 10/02/20 8:15
  end   : 16:15
  participants: "Peter Stein", "Paula Schmidt", "Tina Berg (Personal)"
  weekly
  break {
    start : 11:30
    end   : 12:00
  }
}
```

Exercise 1.4: Types and Expressions (4 Points)

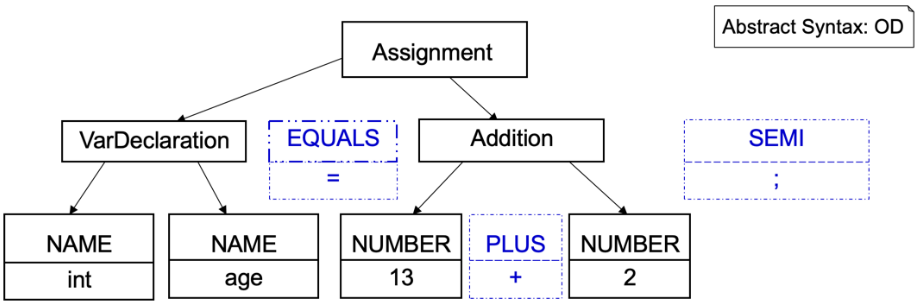
a) Draw the AST structure of the grammars MCBasicTypes[1] and MCCollectionTypes[2]. Use the following graphical syntax:



b) Draw the tree structure of AST nodes that is created if this Expression is parsed by an empty grammar that extends CommonExpressions[3] and MCCommonLiterals[4]:

```
((4 > 17) || (call(25) + 3 != 7)) && !(a || false)
```

Use the graphical notation known from the lecture (slide 4 of the AST chapter):



As a solution of this Exercise sheet, zip and hand in your project without the generated target. Make sure the generated files are reproducible by just running `mvn install`. **Not working projects will not gain you any points**.

[1] https://github.com/MontiCore/monticore/blob/master/monticore-grammar/src/main/grammars/de/monticore/types/MCBasicTypes.mc4

[2] https://github.com/MontiCore/monticore/blob/master/monticore-grammar/src/main/grammars/de/monticore/types/MCCollectionTypes.mc4

[3] https://github.com/MontiCore/monticore/blob/master/monticore-grammar/src/main/grammars/de/monticore/expressions/CommonExpressions.mc4

[4] https://github.com/MontiCore/monticore/blob/master/monticore-grammar/src/main/grammars/de/monticore/literals/MCCommonLiterals.mc4