Universiteit van Amsterdam

Programmeertalen

Schrijfopdracht

L. Thomas van Binsbergen

Case study: eFLINT

Deadline: Vrijdag, Week 3

Practical information

Before answering questions: read through the whole assignment and discuss the resources (below) within your group.

Make sure you include your names, student numbers and group number in your submission. This assignment is executed in a group of three. Only submit one solution per group. Groups are self-registered on Canvas. You can write your answers in any medium you like (text editor, markdown editor, LaTeX/Overleaf, Word, etc.) as long as you **submit a pdf document**. If in your answers you wish to include code fragments, then working in LaTeX, markdown, or plain text are recommended. Write specific and concise answers and include references when citing or quoting outside material. The questions with point indicators sum up to 80 marks out of a maximum of a 100. At least some answers to 'bonus points' questions are therefore required to score above an 8.0 for this assignment.

Introduction

The eFLINT language is a domain-specific language designed for the dual purposes of describing norms from a variety of sources – such as laws, regulations, contracts and organisational policies – and (automatically) assessing cases for compliance with the described norms. One can say that eFLINT actually consists of two languages working tightly together, with one part used for describing norms and the other for describing cases. From our perspective with the Programmeertalen course, the language is interesting in that it combines concepts from both declarative and imperative programming. The purpose of this assignment is to investigate this aspect of the language in more detail. To do so, you will use the following resources:

- The original paper introducing the language, found here [2]. Optional further reading can be found here [1]
- Notebooks containing documentation on features of the languages found here: `https://gitlab.com/eflint/haskell-implementation/-/tree/master/notebooks`. The main file to study is `1_tutorial_voting.ipynb`. To learn more about notebooks, search for 'Jupyter Notebooks'.
- The following basic web-interface for running test programs. Consider in particular the 'Help with Homework' example that is also discussed in [2].
  `http://grotius.uvalight.net/eflintonline/index.php`
- To get a feel for the language, you can install the `eflint-repl` executable made available by the `haskell-implementation` project in which the notebooks are found. This is not required for the assignment.

Declarative characteristics

During the lecture it was discussed that in some declarative programming languages *relations* are described and manipulated, with the logical programming language Prolog and database languages MySQL as important examples. These languages have in common that they have:

1. means to define the structure of relations,
2. means to add/remove elements to/from relations, and
3. means for inferring additional elements (logical inference).

Q1 (30 points) Describe the parts of the eFLINT language used for defining relations (1 above), manipulating relations (2) and performing inference (3). Do not only list the constructs; describe how they are used to program with relations.

Q2 (5 points) In eFLINT you can write `?<BOOL-EXPR>.`, i.e. a question mark followed by a Boolean expression followed by a full stop. Using the terminology of mathematical relations, logic programming, or database programming, what is this construct used for?

Q3 (bonus points) What (non-syntactical) differences do you see between the language constructs discussed in the answers to Q1 and Q2 and similar constructs in Prolog or MySQL? Consider, for example, expressive power, or behaviour when executed. You can also compare with other logic programming or database languages if you prefer. Be specific.

Imperative characteristics

In imperative programming, the programmer gives instructions (statements) to the machine that can have several effects, e.g. causing the machine to read input (standard input channel, command line arguments, file system,...), write output (standard output channel, file system, database,...) or change the contents of memory.

Q4 (30 points) Describe the parts of the eFLINT language that can be classified as imperative instructions. Explain how you decided that these are indeed the instructions of the language. For each kind of instruction, explain the effects these instructions can have.

Q5 (bonus points) What would you say is the *state* manipulated by an eFLINT program? How does it change over time? How does the manipulation of state help with assessing a normative (legal) case for compliance?

Semantic analysis

Q6 (15 points) Explain in detail when a duty is considered violated and in which ways a duty can become violated, for example the `help-with-homework` duty of the 'Help with Homework' example. In your explanation you can give and refer to examples. However, your explanation needs to be valid for all duties defined in the language.

Q7 (bonus points) Argue whether the language is statically or dynamically typed. How did you determine this?

Q8 (bonus points) Write a C function that implements the `ask-for-help` act from the 'Help with Homework' example. The goal is that when this function is called with some arguments, the function behaves like when an instance of the `ask-for-help` type is triggered. Alternatively: use Haskell, Prolog, MySQL or pseudo-code instead of C.

Referenties

[1] van Binsbergen, L. T., Kebede, M. G., Baugh, J., van Engers, T., and van Vuurden, D. G. Dynamic generation of access control policies from social policies. *Procedia Computer Science 198* (November 2022), 140–147.

[2] van Binsbergen, L. T., Liu, L., van Doesburg, R., and van Engers, T. eFLINT: A Domain-Specific Language for Executable Norm Specifications. In *Proceedings of the 19th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences* (2020), GPCE 2020, ACM, pp. 124—136.