An Online Logic Programming Development Environment

Christian Reotutar,[1] Mbathio Diagne,[2] Evgenii Balai,[3] Edward Wertz,[3]
Peter Lee,[4] Shao-Lon Yeh, [5] Yuanlin Zhang[3]
[1]Department of Computer Science, Johns Hopkins University, USA
[2]Department of Mathematics, Minneapolis Community and Technical College, USA
[3]Department of Computer Science, Texas Tech University, USA
[4]Department of EECS, University of California, Berkeley, USA
[5]Lubbock High School, Lubbock, Texas, USA
[1]creotut1@jhu.edu, [2]rv6793jp@go.minneapolis.edu, [3]{evgenii.balai, edward.wertz, y.zhang}@ttu.edu,
[4]peter.lee@berkeley.edu, [5]shaolonyeh@yahoo.com

Abstract

Recent progress in logic programming, particularly answer set programming, has enabled us to teach it to undergraduate and high school students. We developed an online answer set programming environment with simple interface and self contained file system. It is expected to make the teaching of answer set programming more effective and help us to reach more students.

Introduction

Answer Set Programming (ASP) (Gelfond and Kahl 2014) becomes a dominating language in knowledge representation community because it has offered elegant and effective solutions not only to the classical Artificial Intelligence Problems but also many challenging application problems. Thanks to its simplicity and clarity in both informal and formal semantics, Answer Set Programming provides natural modeling of many problems.

ASP has been taught to undergraduate students, in the course of Artificial Intelligence, at Texas Tech for more than a decade. We believe ASP has become mature enough to be a language for us to introduce programming and problem solving to high school students. We had offered 4 sessions to students at New Deal High School and a three week long ASP course to high school students involved in TexPREP program (http://www.math.ttu.edu/texprep/). In our teaching practice, we found that ASP is well accepted by the students and the students were able to focus on the problem solving, instead of the language itself. The students were able to write programs to answer questions about the relationship (e.g., parent, ancestor) among family members and to find solutions for Sudoku problems.

However, we have some major issues with the use of the existing tools: installation of the tools to computers at lab or home is complex and the existing tools are sensitive to local settings of a computer. As a result, the flow of the class teaching were often interrupted by the problems associated with the use of the tools. The aim of the work reported here is to develop an easy to use ASP tool for teaching ASP.

Background

Answer set programming is variation of Logic Programming, whose semantics is based on stable models (Gelfond and Kahl 2014). To show the ease of representation of some interesting problems, we consider the map coloring problem here. We have a few states such as Texas, New Mexico and Colorado. We use `neigbor(X,Y)` to denote that state `X` is a neighbor of state `Y`. We can represent our knowledge of the connectedness of states as fact such as `neighbor(texas, newMexico).` We noted that X is a neighbor of Y if Y is a neighbor X. This knowledge is represented as an ASP rule

```
neighbor(X, Y) :- neighbor(Y, X).
```

where `:-` is read as if. To express the intention that we would like to assign a color for `r, g, b` to a state, we introduce a relation $colorOf(S, C)$ which denotes that the color of state S is C. The knowledge that a color is assigned to a state is represented as an ASP rule

```
color(S,r) | color(S,g) | color(S,b).
```

where `|` is read as or. The rule reads, for any state S, S has a color of `r`, `g`, or `b`.

To express the fact that no two neighbors have the same color, we use the rule

```
¬color(S2, C1) :- color(S1, C1),
        neighbor(S1, S2), S1 != S2.
```

where ¬ is the classical negation. The rules says for any distinct states `S1` and `S2`, if the color of `S1` is `C1`, the color of `S2` is not `C1`, i.e., its color is different from that of `S1`. Now we finish the full ASP program for map coloring. As one can see, the rule is close to the specification of the problem in English. A stable model of the program is a set of relations that is believed by a rational agent in terms of the program. For example, a rational agent has to believe `neighbor(texas,newMexico)` because it is a fact. An example of a stable model the program is tt {color(texas,r), color(newMexico,g), color(colorado,b), ...} where we list only color information while ignoring the neighbor relations among the three states. We can prove that the mapping coloring problem for these states has a solution if and only if the program has a stable model.

As ASP has been applied to more and more problems, the importance of software development tools for ASP has

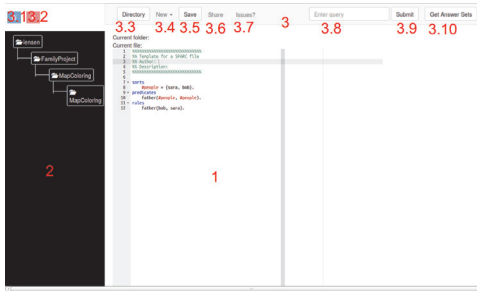

Figure 1: User Interface of the System (the red numbers indicate the areas/components in the interface)

been realized recently. Some integrated development environment (IDE) tools, e.g., ASPIDE(Febbraro, Reale, and Ricca 2011), have been developed. They provide a graphical user interface for users to carry out a sequence of task from editing the ASP program to debugging the program, easing the use of ASP significantly. However, we noted that the target audience of these tools are experienced software developers. The tools also suffer from the issues mentioned in the introduction section.

Our Solution

By studying the issues in our teaching practice, we realize the following obstacles.1) The existing tools are standalone software. It is expensive to maintain those tools during and outside the class. 2) The use of the environment needs the students to have some knowledge of the directory structure and how it is connected to the development tools. 3) Complex user interface packed with many functions distracts the attention of students from the key ASP concepts and problem solving. 4) Sharing of the programs (e.g., submission to the instructors) is challenging.

To overcome these obstacles, developed an online ASP development environment, which can get rid of installation and maintenance once the web application is set up. It will also get rid of most of the failures due to the settings of a local computer. It will also make the environment accessible anywhere and anytime. We remove the connection of local directory structure and the development environment by providing a file system inside our online application. This online file system also makes it easy for students to submit their programs to instructors by simply sharing them. To make the user interface accessible to novice users, we offered a new interface which is significantly different from the existing tools.

System Design and Implementation

In this section we present the design and implementation of the system which is avail at http://goo.gl/ukSZET.

1. User Interface (UI). The largest UI component is the editor with a syntax highlighting features (1 in Figure 1). One can edit SPARC program directly here. To ask queries to the program, one can type a query in text box (3.8) and then press the button (3.9) to execute it. To see the stable models of the program, click the button (3.10). If a user wants to save the program, they needs to log-in (by clicking button to the right of 3.10 not visible in the figure) to the system. They can then save the program (3.5) to a file (on the server running the online environment). To organize files, the user can also create folders (3.4). The file system (files/folders) can be navigated in the left panel (2) which can be displayed or hidden by pressing button (3.3). One can share their files/folders with another user of the application by clicking button (3.6). A temporary button (3.7) is provided for users to send feedback to the developers.

2. Implementation. The implementation includes three components: front end (FE) in JavaScript, processing unit (PU) in PHP, and back end (BE) (SPARC solver (Balai, Gelfond, and Zhang 2013) and MySQL database system). The structure of the file system is implemented using database tables in BE and each file is saved as a file on the server side. PU will pass to FE the needed folder structure and files, and then manage the files (through MySQL and file system of the server side) as requested by FE. When obtaining a request for query or finding a stable model from the FE, PU will call the SPARC solver in BE with the right program and then parse and return the results from SPARC solver to FE. The sharing is managed by the sharing information in the relevant database tables.

Conclusion

Most existing programming environments are proved useful for experienced programmers, but it is still challenging to use them in teaching students who are novice in programming. Our online environment, with a carefully designed simple interface and a self contained file system, provided easy access and sharing, reduced the learning curve, and removed the installation and maintenance expenses, by our experience of using it in our teaching in Fall 2015. We take it as an enabler to teach ASP to more students including high school ones.

Acknowledgment

We thank Michael Gelfond for his input on the interface design. This work is partially supported by NSF grant IIS-1018031 and CNS-1359359.

References

Balai, E.; Gelfond, M.; and Zhang, Y. 2013. Towards answer set programming with sorts. In *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings*, 135–147.

Febbraro, O.; Reale, K.; and Ricca, F. 2011. ASPIDE: integrated development environment for answer set programming. In *Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings*, 317–330.

Gelfond, M., and Kahl, Y. 2014. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents*. Cambridge University Press.