Online SPARC for Drawing and Animation

Elias Marcopoulos,[1] Maede Rayatidamavandi,[2] Crisel Suarez,[3] Yuanlin Zhang[4]
[1]Department of Computer Science, Tufts University, USA
[2]Department of Computer Science, Texas Tech University, USA
[3]Department of Mathematics, St. Edward's University, USA
[4]Department of Computer Science, Texas Tech University, USA
[1]emarcopoulos@gmail.com, [2]maede.rayati@gmail.com,
[3]csuarez@stedwards.edu, [4]y.zhang@ttu.edu

Abstract

We developed a method to draw and animate using SPARC, a logic programming system, and an online environment to support this method. Particularly, we introduce two predicates: one for drawing and one for animation. By our method, programmers will write a SPARC program, using our introduced predicates, to specify their drawing or animation. The drawing or animation will then be rendered upon executing the program with our system. In fact, our online system provides an environment where the programmers can easily edit and execute their programs.

Introduction

We have developed an online environment to produce drawings and animations using a logical programming language called SPARC. The online environment is available at *http://goo.gl/ukSZET*.

SPARC is a special instance of Answer Set Programming (ASP) (Gelfond and Kahl 2014). ASP is a recent successful development in Logic Programming (Kowalski 2014). It is now fully declarative and gets rid of procedural features of classical Logic Programming systems such as PROLOG. The procedural features are taken as the source of misconceptions in students' learning of Logic Programming (Mendelsohn, Green, and Brna 1990). SPARC is designed to further facilitate the teaching of logic programming by introducing sorts (or types) which simplify the difficult programming concept of *domain variables* in classical ASP systems such as Clingo (Gebser et al. 2011) and help programmers to identify errors early thanks to sort information. Initial experiment of teaching SPARC to high school students is promising (Reyes et al. 2016).

It is observed that multimedia and visualization play a positive role in promoting students' learning (Guzdial 2001; Clark et al. 2009). By introducing drawing and animation to SPARC, we expect to provide new opportunities for students to present their solutions to problems in a more visually straightforward and exciting manner (instead of the answer sets which are simply a set of literals). We integrate our program for rendering the drawing and animation for SPARC programs into the SPARC online development environment (Reotutar et al. 2016). The use of an online system is much easier than a standalone software, which is necessary for teaching K-12 or general undergraduate students.

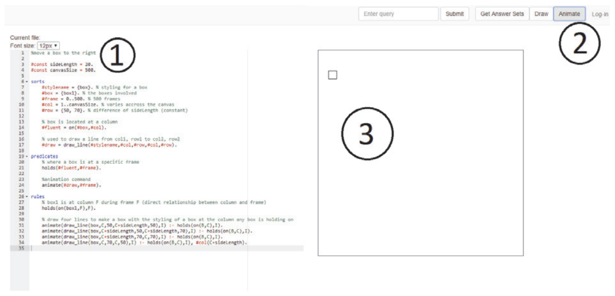
Drawing and Animation Design

To allow programmers to create drawings and animations, we simply design two predicates, called *display predicates*: one for drawing and one for animation. The atoms using these predicates are called *display atoms*.

Drawing. A *drawing predicate* is of the form: `draw(`c`)` where c is called a *drawing command*. Intuitively the atom containing this predicate draws texts and graphics as instructed by the command c. By drawing a picture, we mean a *shape* is drawn with a *style*. A *shape* is a geometric line or curve. A *style* specifies the physical visual properties of the shape it is applied to. For example, one visual property is color. Note the origin of the coordinate system is at the top left corner of the canvas. Here is a an example of drawing a red line from point $(0, 0)$ to $(2, 2)$. First, we introduce a style name `redline` and associate it to the red color by the *style command* `line_color(redline, red)`. With this defined style we then draw the red line by the *shape command* `draw_line(redline, 0, 0, 2, 2)`. Style commands and shape commands form all drawing commands. The SPARC program rules to draw the given line are

```
  draw(line_color(redline, red)).
  draw(draw_line(redline, 0, 0, 2, 2)).
```

Animation. A frame, or drawing/picture, is a critical concept for animation. When a sequence of frames (whose content is normally relevant) is shown on the screen in rapid succession (usually 24, 25, 30, or 60 frames per second), a fluid animation is seemingly created. To design an animation, a designer will specify the drawing for each frame. Given that the order of frames matters, we give a frame a value equal to its index in the given sequence. We introduce the *animate predicate* `animate(`c, i`)` which indicates a desire to draw a picture at the i^{th} frame using drawing command c and i starts from 0. The frames will be shown on the screen at a rate of 60 frames per second, and the i^{th} frame will be showed at $(i * 1/60)$ second from the start of the animation for a duration of $1/60$ second.



```
%move a box to the right

#const sideLength = 20.
#const canvasSize = 500.

sorts
    #stylename = {box}. % styling for a box
    #box = {box1}. % the boxes involved
    #frame = 0..500. % 500 frames
    #col = 1..canvasSize. % varies accross the canvas
    #row = {50, 70}. % difference of sideLength (constant)

    % box is located at a column
    #fluent = on(#box,#col).

    % used to draw a line from col1, row1 to col2, row2
    #draw = draw_line(#stylename,#col,#row,#col,#row).

predicates
    % where a box is at a specific frame
    holds(#fluent,#frame).

    %animation command
    animate(#draw,#frame).

rules
    % box1 is at column F during frame F (direct relationship between column and frame)
    holds(on(box1,F),F).

    % draw four lines to make a box with the styling of a box at the column any box is holding on
    animate(draw_line(box,C,50,C+sideLength,50),I) :- holds(on(B,C),I).
    animate(draw_line(box,C+sideLength,50,C+sideLength,70),I) :- holds(on(B,C),I).
    animate(draw_line(box,C+sideLength,70,C,70),I) :- holds(on(B,C),I).
    animate(draw_line(box,C,70,C,50),I) :- holds(on(B,C),I), #col(C+sideLength).
```

Figure 1: Online SPARC Environment

As an example, we would like to elaborate on an animation where a red box (with side length of 10 pixels) moves from the point $(1, 70)$ to $(200, 70)$. We will create 200 frames with the box (whose bottom left corner is) at point $(i + 1, 70)$ in i^{th} frame. In any frame I, we specify the drawing styling $redline$:

```
animate(line_color(redline, red), I).
```

To make a box at I^{th} frame, we need to draw, using style `redline`, its four sides: bottom - $(I + 1, 70)$ to $(I + 1 + 10, 70)$, left - $(I + 1, 70)$ to $(I + 1, 60)$, top - $(I + 1, 60)$ to $(I+1+10, 60)$ and right - $(I+1+10, 60)$ to $(I+1+10, 70)$. Hence we have the rules

```
animate(draw_line(redline,I+1,70,I+11,70),I).
animate(draw_line(redline,I+1,70,I+1,60),I).
animate(draw_line(redline,I+1,60,I+11,60),I).
animate(draw_line(redline,I+11,60,I+11,70),I).
```

Algorithm, Implementation and Environment

The input to the main algorithm is a SPARC program P. The output is an HTML5 program containing a canvas which will be rendered by the browser. The algorithm finds an answer set (i.e., all atoms that are true under the program by stable model semantics (Gelfond and Kahl 2014)), extracts all display atoms, and generates an HTML5 program that uses canvas to set the drawing style properly according to the style atoms for the i^{th} frame and then renders all shape commands specified by the animate atoms for the i^{th} frame. The drawing commands inside the display atoms will be rendered for every frame. (An optimization is made to reduce repeated rendering efforts.) We integrated our algorithm into the existing online SPARC environment. The interface of our environment is shown in Figure 1. The programmer can edit a SPARC program in area 1 (as shown in the figure). Only after clicking the Execute button (area 2) will the drawing/animation be shown in area 3. Example SPARC programs with drawing and animation can be found at https://goo.gl/nLD4LD.

Discussion and Conclusion

Our work is based on (Cliffe et al. 2008) which first introduced a design of the display predicates and rendered the drawing and animation using a standalone program ASPviz. The first main difference is that our design of the animate predicate has a straightforward meaning, which is important for both teaching and system design, while their design is more complex. The second is that our system is an online environment, whose convenience enables its using in teaching for high school and general undergraduate students. Finally, our host language is SPARC, which is arguably more suitable for teaching than ASPviz's host language.

It is noted that thanks to ASP/SPARC rules, one can define more abstract and easy to use drawing/animation "commands." With the new drawing and animation features, students can not only solve problems such as Sudoku and AI problems, but can also present the results in vivid and straightforward drawings and animations. We hope the new environment will inspire more interest in Logic Programming, AI, and computer science in general, as well as provide a more effective learning environment.

Acknowledgments

The work of Marcopoulos, Suarez and Zhang were partially supported by National Science Foundation (grant# CNS-1359359). We thank Michael Gelfond and Yinan Zhang for their input and help. We thank Texas Tech University (TTU) and the graduate students at KRLab of TTU.

References

Clark, D.; Nelson, B.; Sengupta, P.; and DAngelo, C. 2009. Rethinking science learning through digital games and simulations: Genres, examples, and evidence. In *Learning science: Computer games, simulations, and education workshop sponsored by the National Academy of Sciences, Washington, DC*.

Cliffe, O.; De Vos, M.; Brain, M.; and Padget, J. 2008. Aspviz: Declarative visualisation and animation using answer set programming. In *International Conference on Logic Programming*, 724–728. Springer.

Gebser, M.; Kaufmann, B.; Kaminski, R.; Ostrowski, M.; Schaub, T.; and Schneider, M. 2011. Potassco: The potsdam answer set solving collection. *Ai Communications* 24(2):107–124.

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

Guzdial, M. 2001. Use of collaborative multimedia in computer science classes. In *ACM SIGCSE Bulletin*, volume 33, 17–20. ACM.

Kowalski, R. 2014. Logic programming. *Computational Logic, Volume 9 (Handbook of the History of Logic)*.

Mendelsohn, P.; Green, T.; and Brna, P. 1990. Programming languages in education: The search for an easy start. *Psychology of programming* 175–200.

Reotutar, C.; Diagne, M.; Balai, E.; Wertz, E.; Lee, P.; Yeh, S.-L.; and Zhang, Y. 2016. An online logic programming development environment. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Reyes, M.; Perez, C.; Upchurch, R.; Yuen, T.; and Zhang, Y. 2016. Using declarative programming in an introductory computer science course for high school students. In *Thirtieth AAAI Conference on Artificial Intelligence*.