A Rule Based Approach to teach Mathematics using Animation

Nada Sharaf[1], Slim Abdennadher[1], Thom Frühwirth[2]

[1] The German University in Cairo
[2] University of Ulm
{nada.hamed, slim.abdennadher}@guc.edu.eg, thom.fruehwirth@uni-ulm.de

Abstract. There are different available methodologies for teaching mathematics to children. Teachers use different approaches. Some of the exiting approaches include engaging students with different activities and games. Computer games/tools have been proven effective with teaching Maths. The aim of the paper is to provide teachers with no background in Computer Science with a utility that enables them to build their own games. In this way, teachers will be able to customize games according to the principles they want their students to learn.

Keywords: Constraint Handling Rules, Maths, Learning, Program Animation, Visualization

1 Introduction

One of the recently introduced activities for teaching mathematics is using computer games [1,2]. Students have proven to have positive attitudes towards games involving maths [3]. Such approach encourage active learning [4]. Such tools have been proven to be effective in enhancing learning of complex content as well [5].

The provided tools are however static ones. Teachers do not have the option to customize the games in any way. For example, the appearance cannot be changed. Users will have to stick to the look provided by the original programmers of the tools. In addition, teachers cannot customize the mathematical concepts tackled by the tools. They would thus have to use different games/tools in case they need to tackle more than one concept. The work in this paper aims at overcoming this drawback. It introduces a new rule-based approach for generating different interactive customizable games. Through the offered tool, teachers are able to define the mathematical concepts students should practice. They are then able to state how numbers should be visualized.

The tool makes use of the recently introduced annotation rules for animating Constraint Handling Rules (CHR) programs [6]. Such rules were able to embed visualization features into CHR programs. With the new extension, CHR programs were animated while execution. The tool used source-to-source transformation to eliminate the need of changing the CHR compiler.

The tool introduced in the paper generates CHR programs representing the different mathematical concepts entered by the teacher. Annotation rules are

then utilized to visualize the execution of the program using the inputs of the teacher.

The paper is organized as follows: Section 2 introduces CHR. In Section 3, annotation rules are discussed in more details. Section 4 introduces the different features offered to teachers. Section 5 shows how students can use the platform. Finally, conclusions and directions to future work are offered.

2 Constraint Handling Rules

CHR [7,8,9,10] was initially introduced for writing constraints solvers. However, over the years it has been used as a general purpose language. CHR programs consist of different rules that rewrite constraints in the constraint store until a fixed point is reached. A CHR rule has a head and a body and an optional guard. A rule is only applied if the constraint store contains constraints matching the head constraints and if the guard is satisfied. For example the below "simplification rule" is able to sort a list of numbers. Each element in the list is represented by the CHR constraint `list(Index,Value)`.

```
list(I1,V1),list(I2,V2)<=>I1<I2,V1>V2|list(I2,V1),list(I1,V2).
```

The rule is applied on any two elements that are not sorted. The elements are then swapped with respect to each other by removing the head constraints from the store and adding the constraints in the body. In order for the rule to be applied, two list constraints have to be in the store. The two constraints have to satisfy the guard as well. On successive applications of the rule, all elements are sorted. Propagation rules, on the other hand, do not remove the head constraints from the constraint store. They only add the body constraints such as the transitivity rule:

```
leq(A,B),leq(B,C)==>leq(A,C).
```

The last, and more general, type is the simpagation rule. A simpagation rule has two types of head constraints separated by a backslash: "\". On executing a simpagation rule, the constraints before the backslash are kept and the ones after are removed. For example the rule `min(A)\min(B)<=>A<B|true` compares two constraints and keeps only the one having the lower number. Thus on applying this rule successively, the only constraint remaining is the one with the lowest number.

3 Annotation Rules for Animating CHR Programs

With CHR becoming a general purpose language, the need of tracing tools aroused. In [11], a tracing utility for CHR was added. It was able to show at each step of the execution, the constraint store and which rules were being applied. However, since CHR is used with different types of algorithms (such as sorting, tree and graph algorithms), an algorithm animation tool was required. In order to keep the platform a generic one, visual annotation rules were added [6]. The idea is that every CHR constraint was linked to a visual object. CHR

rules operate on constraints, adding and removing them. Each time a new constraint is added to the store, its corresponding visual object(s) (if any) is added. Users are thus supplied with an interface to mark the interesting constraints. Such constraints affect the visualization and are linked to visual objects. For example, in the sorting program shown in Section 2, the interesting constraint is `list/2`. Every list constraint could be visualized as a bar where the height of the bar is a factor of the "value" of the list element. The x-position of the bar is a factor of the "index" of the list element. The whole list is visualized to the user in this case. When a list constraint is added/removed, its corresponding bar is added/removed thus animating the algorithm executed. To keep the system generic, the scripting tool Jawaa[1] was used. Jawaa offers users with a wide range of visual objects.

4 Teacher Module

In this section, the "teacher module" is introduced. This module is used by teachers to specify the mathematical concepts students should learn and the appearance of the output game. Figure 1 shows the first screen teachers get. They have the option to define a "simple rule" and a "Rule with Steps". A simple rule is a rule computed through one step. Teachers can also define a rule with several cases/steps.

Simple Rules In the case of simple rules, teachers have to define the input(s) and output. Figure 2a shows the view teachers get once they decide to add a simple rule. The name of the rule is editable. As seen in Figure 2b, it was changed to sum. Users can enter any number of inputs. An input could be a variable name or an actual value as seen in Figure 2b. The output could also be a value or an expression as shown in Figure 2c.

Rules with Steps In this case instead of only defining the expression for the output, teachers define steps. Except for the first step each step takes the output of the previous step as one of its inputs. There is an upper limit to the number of steps that could be performed. For space issues, the paper will focus only on the case of simple rules since the core principles for generating the animations in both cases are the same.

4.1 Defining Animations

After teachers define the mathematical rule that students should practice, they can define how the quizzes students get look like. They can first choose color or an image for the background. They can also specify how numbers should appear. The idea is that each number n could be represented by a visual object or n visual objects. Teachers can customize what the objects are. Objects could be simple shapes (provided by Jawaa) such as circles, rectangles, etc. Objects could also be linked to pictures to match a specific theme. Teachers get the window shown

[1] `http://www.cs.duke.edu/csed/jawaa2/`

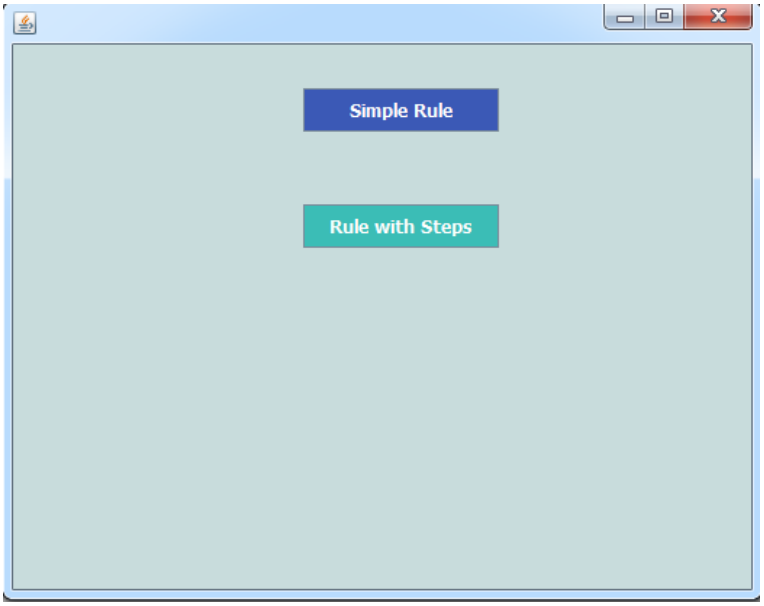
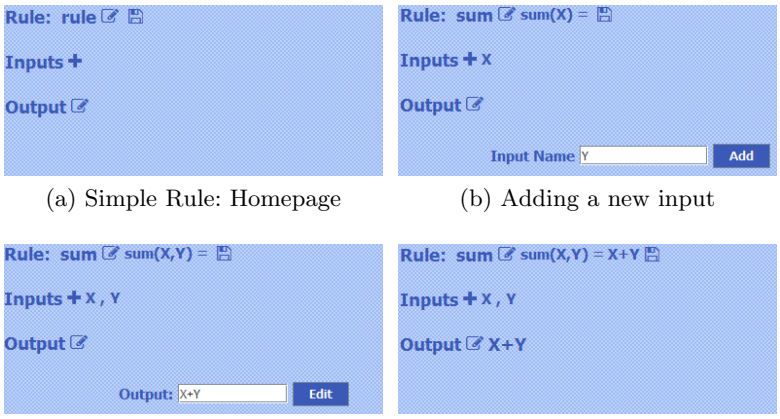


Fig. 1: Teacher Module Welcome Page



(a) Simple Rule: Homepage

(b) Adding a new input

(c) Editing Output

(d) Summation Rule defined

Fig. 2: Simple Rules: Inputs and Outputs.

in Figure 3 where they can link a number to its corresponding visual object to produce the required annotation rules. Once the teacher chooses an object, the panel gets populated with the corresponding parameters that have to be filled. In order to link a number to a number of objects, the teacher should choose

to connect it with the visual object "nObjects". *nObjects* is an abstraction of grouping several visual objects together.

For the example shown in the Figure 3, it was required that a number `X` gets associated with `X` different objects. The number was thus linked with "nObjects". The teacher should then choose which type of visual object should be generated N times for each number. The object "imageobject" was used in this case. An *imageobject* is an actual image with the extension "jpg" or "png". For each imageobject users have to specify where the image should be shown (x and y coordinates) in addition to the location of the image (path). In general, each parameter could have one of the following values:

1. a constant e.g. 30, red, etc
2. the built-in function $valueOf(X)$ representing the value of the number X.
3. the keyword $valueOf(N)$ used in the case of nObjects to represent the varying number. For example the first generated object would have an N=0, the second would have an N=1 , etc.

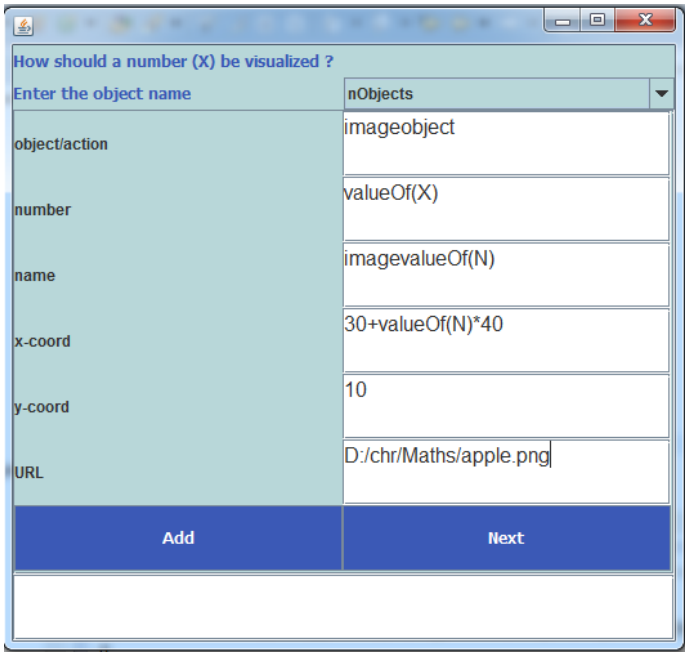


Fig. 3: Link a Number to a Visual Object

In the previous example the x-coordinate of each shown imageobject was set to $30 + valueOf(N) \times 40$. Thus for the first image shown the x-coordinate will be $30+0\times40$ which is 40. The second image will have an x-coordinate of $30+1\times40$ or 70, etc. A user can also associate a number through more than one rule. For

example, a number could be associated with a colored circle and a text object. Thus more than one object are shown for the same number.

Once the teacher defines the needed annotation rule, they move to the next (optional) step. In this step, teachers can define any number of constraints on the input numbers students will get. For example teachers can add constraints for an input to be a one-digit number (i.e. < 10 and ≥ 0). Constraints can also link more than one input together (e.g. $X < Y$). Figure 4 shows an example where every input has more than one associated constraint. The available constraints are $<$, $>$, $\leq$, $\geq$, $=$, $!=$. Teachers also choose lower and upper bounds for the generated numbers.

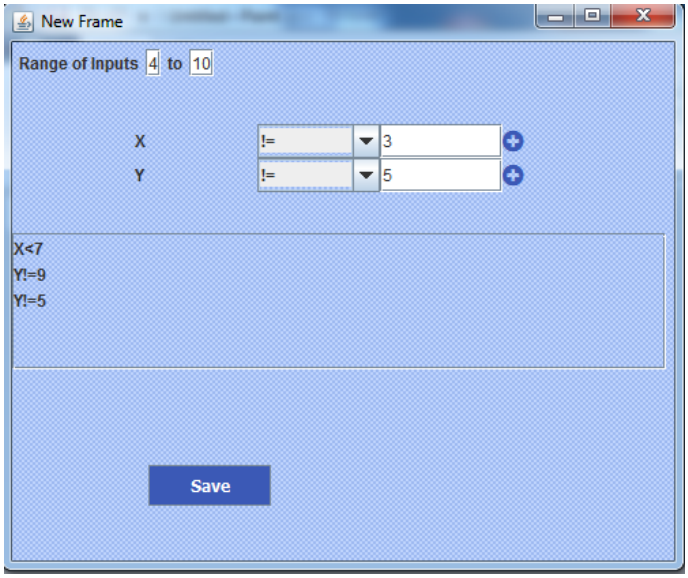


Fig. 4: Restricting generated numbers

4.2 Translation to CHR Programs

Every simple rule named rule name with inputs $X_0, \ldots, X_{N-1}$ is represented with the CHR constraint `rule(rule name, N)`. Inputs are represented separately through the constraints `input(Rule_name,Var,Index)` The output of the rule contains in most of the cases an expression to be evaluated. The evaluated output is stored inside the constraint `output(Rule_name,Output)`. Thus such a simple rule is represented by the following CHR rule:

```
:-chr_constraint rule/2, input/3, output/2.
rule(Rule_name,N),input(Rule_name,X0,0),...,input(Rule_name,XN,N),
                        <=> Output is Expression, output(Rule_name,Output).
```

The generation of the CHR file is automatically done. Thus, the teacher does not have to be aware of CHR to use the system. The produced file is transformed using the CHRAnimation tool to be able to produce the required visual objects while execution.

5 Student Module

Once users start to play, the background is set to the background chosen by the teacher. It could be just a color or an image. Afterwards, the random generator is used to generate numbers fulfilling the needed constraints. Once the numbers are generated the CHR file produced in Section 4.2 is queried. The aim of querying the CHR file is to:

- generate the correct output to be able to compare the answer of the student to the correct one.
- represent the inputs and output as CHR constraints activating the animation.

Every input is associated with the constraint `input/3`. Every time, such a constraint is added, its corresponding visual object(s) is added. For instance, in the previous example, every input with value `X` is associated with `X` pictures showing an "apple". Thus every time a constraint for an input is added, CHRAnimation adds the corresponding visual objects to the animation frame resulting in the window shown in Figure 5a showing the two input numbers (2 and 4)[1].

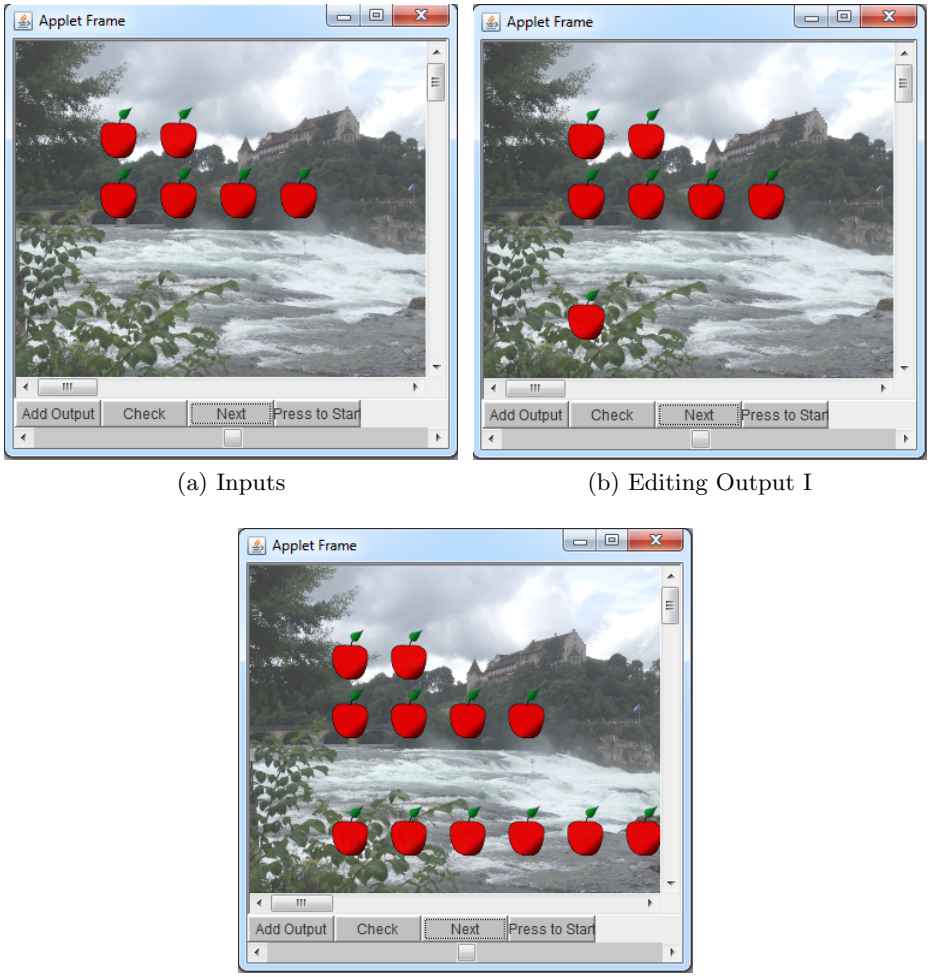
The student can then start to add the suggested output. Every time the student presses "Add", the output is incremented. Since the output is a number, it is visualized in the same way. Figure 5b shows the window after pressing the button one time. The output is thus now visualized with one apple. Figure 5c shows the window after setting the output to 6. In this case, six apples are shown. At any point, the student can "check" whether the current output is correct or not. They get the corresponding message in each case.

5.1 Another Quiz

Another option for producing interesting interactive animations is to :

1. Link every input number with a normal Jawaa circular node. The text inside the node is its value. Its background is blue.
2. Link the output with a random number of nObjects displaying a group of nodes. Each node is placed in a random position. The text inside each node is also a random number. CHRAnimation has the keyword "Random" that could be utilized in this case. The background of those nodes is green.

[1] The y-coordinate specified by the teacher is automatically multiplied by the index of the input to have each input on separate line



(a) Inputs

(b) Editing Output I

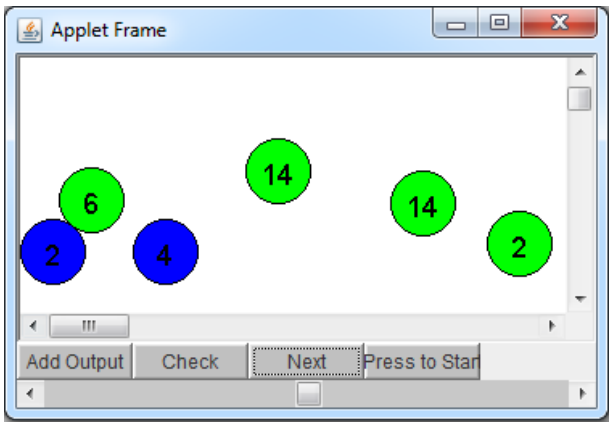
(c) Editing Output II

Fig. 5: Quiz 1

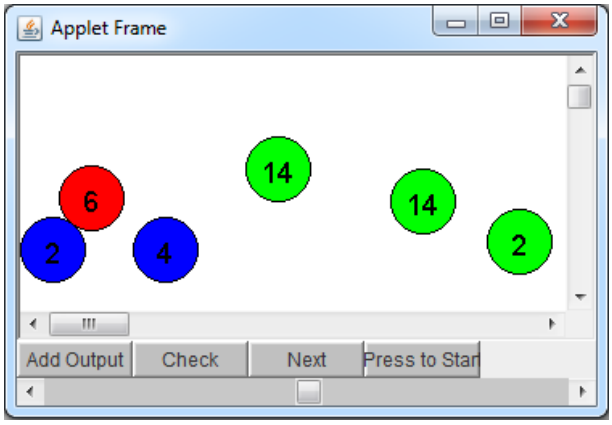
3. Link the output with a Jawaa circular node with the name (jawaanodecout) displaying the actual output of the rule. It is also placed in a random position. Its background is also green.
4. Add an annotation rule linking the output constraint with an `onclick` command for the object `jawaanodecout`. Once it is clicked, the `changeParam` command is activated changing its color to red.

Note that through the GUI, users do not have to know any details regarding the syntax of the annotation rules. Once the generated CHR file is queried two blue nodes representing the two inputs are shown. In addition, a group of green

nodes are shown. One of them only represents the output. Once the user clicks on the node representing the output value, its color changes to red. If the user clicks on any other node, nothing happens. Figure 6a shows the initial setup with the randomly placed nodes. Figure 6b shows the node with the output being highlighted after the user clicked it.



(a) Randomly placed nodes



(b) Highlighted node after clicking

Fig. 6: Quiz 2

6 Conclusion & Future Work

This paper shows how annotation rules could be utilized for generating quizzes to teach Maths. The tool was able to customize the look of the games according to

the inputs of the teachers unlike existing games with static looks and operations (such as: `http://www.iboard.co.uk/iwb/Simple-Addition-Stories-721`). The tool does not need any computer science background. As seen through the examples, annotation rules were able to produce interactive animations that could be used to teach mathematical rules. In the future, different mathematical concepts should be explored and animated. The tool should be linked with different visualization libraries as well. The paper offered a prototype for a proof of concept. In the future, the tool should be extended in a way to handle different kinds of output quizzes in a generic way.

References

1. S. Barab, M. Thomas, T. Dodge, R. Carteaux, and H. Tuzun, "Making learning fun: Quest atlantis, a game without guns," *Educational Technology Research and Development*, vol. 53, no. 1, pp. 86–107, 2005.
2. T. Amon, "Simulations and the future of learning: An innovative (and perhaps revolutionary) approach to e-learning," *Educational Technology & Society*, vol. 7, no. 3, pp. 149–150, 2004.
3. F. Ke, "A case study of computer gaming for math: Engaged learning from gameplay?," *Computers & Education*, vol. 51, no. 4, pp. 1609–1620, 2008.
4. R. Garris, R. Ahlers, and J. E. Driskell, "Games, motivation, and learning: A research and practice model," *Simulation Gaming*, vol. 33, no. 4, pp. 441–467, 2002.
5. K. E. Ricci, E. Salas, and J. A. Cannon-Bowers, "Do Computer-Based Games Facilitate Knowledge Acquisition and Retention?," *Military Psychology*, vol. 8, no. 4, pp. 295–307, 1996.
6. N. Sharaf, S. Abdennadher, and T. W. Frühwirth, "Chranimation: An animation tool for constraint handling rules," in *Logic-Based Program Synthesis and Transformation - 24th International Symposium, LOPSTR 2014.* (M. Proietti and H. Seki, eds.), vol. 8981 of *Lecture Notes in Computer Science*, pp. 92–110, Springer, 2014.
7. T. Frühwirth, "Theory and practice of constraint handling rules, special issue on constraint logic programming," *Journal of Logic Programming*, vol. 37, pp. 95–138, October 1998.
8. T. Frühwirth, *Constraint Handling Rules.* Cambridge University Press, aug 2009.
9. H. Betz, F. Raiser, and T. W. Frühwirth, "A complete and terminating execution model for constraint handling rules," *TPLP*, vol. 10, no. 4-6, pp. 597–610, 2010.
10. T. W. Frühwirth, "Constraint handling rules - what else?," in *Rule Technologies: Foundations, Tools, and Applications - 9th International Symposium, RuleML 2015, Berlin, Germany, August 2-5, 2015, Proceedings* (N. Bassiliades, G. Gottlob, F. Sadri, A. Paschke, and D. Roman, eds.), vol. 9202 of *Lecture Notes in Computer Science*, pp. 13–34, Springer, 2015.
11. S. Abdennadher and N. Sharaf, "Visualization of CHR through source-to-source transformation," in *Technical Communications of the 28th International Conference on Logic Programming, ICLP 2012, September 4-8, 2012, Budapest, Hungary* (A. Dovier and V. S. Costa, eds.), vol. 17 of *LIPIcs*, pp. 109–118, 2012.