Solving the Movie Database Case: An e-Motions based solution

Antonio Moreno-Delgado and Francisco Durán

University of Málaga
{amoreno,duran}@lcc.uma.es

Abstract. TO-DO. Podemos darle dos enfoques: o basado en e-Motions, o basado en la lógica de reescritura y presentamos las soluciones de e-Motions y Maude. En caso de usar la primera también podemos usar las soluciones de Maude, como un añadido en cada tarea. En caso de la segunda podemos poner en cada tarea una subsubsection con e-Motions y otra con Maude.

1 Introduction

TO-DO: what is e-Motions? **TO-DO**: why rewriting logic?

1.1 e-Motions

TO-DO: Introduction to e-Motions rules **TO-DO**: solo usaremos reglas sin tiempo

2 Solution

TO-DO: explicar cómo cada tarea viene dada por la definición de un DSL. En la mayoría de los casos la sintaxis puede ser reutilizada pero no el comportamiento (?)

2.1 Task 1

Task 1 consists in to generate synthetic models (conforming the movie database metamodel [2]) from an input parameter $N \geq 0$. Following an e-Motions based approach, we define the abstract and concrete syntax and the behavior of our so-called *Task 1 DSL*, which takes a model with a parameter N and generate as output a model containing synthetic data to be used as test case.

As it has been introduced in Section 1.1, the abstract syntax of a DSL is given by means of a Ecore metamodel, which is provided in [1] and, in the following, we call it *Movies MM*. However, the *parameter N* concept has to be modeled in some way, since in e-Motions the state[(i)] is just a model. Hence, a new concept call `Parameter` with two Integer attributes `nP` and `nN` (positive and negative

[(i)] con este state me refiero al estado del sistema, de una ejecución

graphs respectively) has been added to Movies MM. This results in a so-called *Movies* MM.*(ii)

For the concrete syntax, Fig. 1 shows how an image has been attached to each concept modeled in the Movies* MM. The behavior of this *Task 1 DSL* is given by means of two in-place rules: `createPositive` and `createNegative`. Figure 2a shows the `createPositive` rule, which takes an object `p` of type *Parameter* with *nP* attribute is greater or equal than 0 and, after the rule application, synthetic data conforming to the Henshin rules [2] are created. Fig. 2b shows the `createNegative` rule, which is analogously defined.

Once the syntax and the behavior of the system has been coded, the user may specify a model, which conforms to *Movies* MM*, containing an object `Parameter` with its two attributes `nP` and `nN` properly set. This model is used as initial model of the execution.

(ii) podríamos referenciar a los trabajos donde esto se hace de forma modular

Maude version Concerned with the performance of e-Motions, we have specified a Maude equivalent version of Task 1. This proposal of Task 1 consists of a object-based Maude specification, which is composed by two main modules: `MOVIES@MM` defining the classes structure and `TASK1` defining the solution. The solution is coded using again two rules: `createPositive` and `createNegative`.(iii) **TO-DO**: poner código con esta solución. One could realized that the Maude version is very much like the e-Motions version. In fact, the former is just the textual version of the latter.

(iii) podriamos ponerlo en un apéndice el código

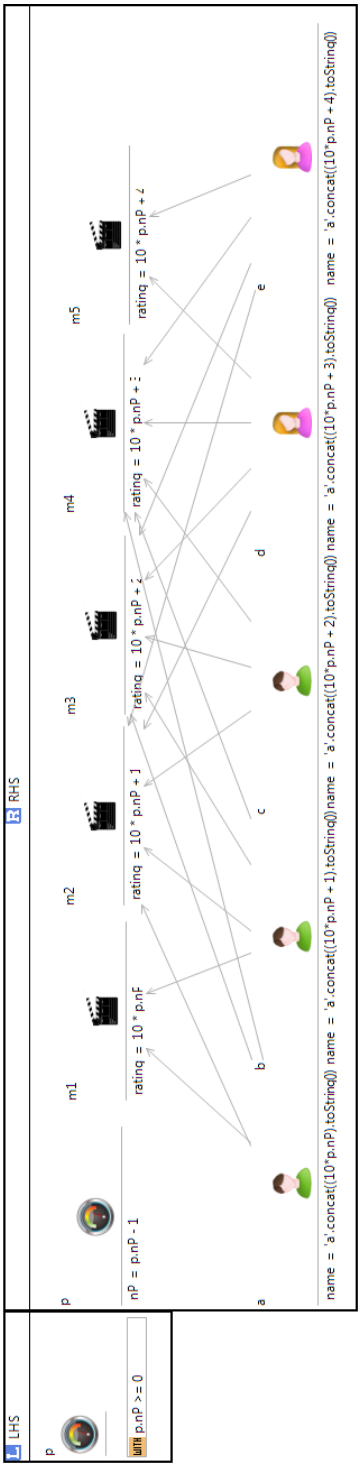
`imgs/concreteSyntax.png`

Fig. 1: Concrete syntax for *Movies* MM.*

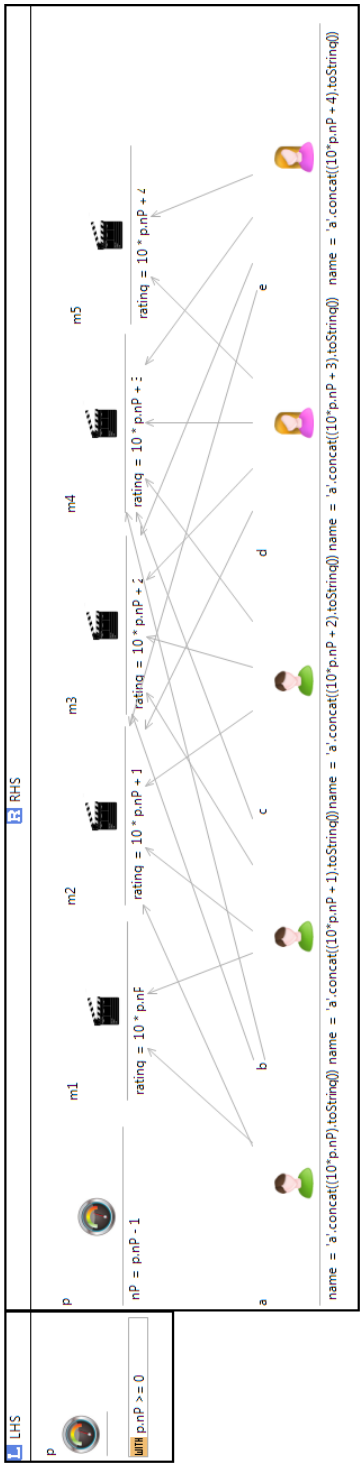
TO-DO: Correctness?
TO-DO: Time tables once we have installed maude in the Ubuntu image

References

1. Horn, T.: IMDB2EMF, `https://github.com/tsdh/imdb2emf`
2. Horn, T., Krause, C., Ticky, M.: The TTC 2014 Movie Database Case, available at TTC14 web site.



(a) The `createPositive` rule.



(b) The `negativePosition` rule.**TO-DO**: change it

Fig. 2: Task 1 rules.