Ecole Polytechnique de Louvain

LINGI2132 - Languages and translators

Assignement 3 - Report

Professor :
Pierre Schaus

Students :
Benoît Baufays 22200900
Julien Colmonts 41630800

Program :
SINF21MS

Academic Year 2013-2014

1 Functionality/specificities of our DSL

Even if our DSL is able to solve any problem, we still added some classes allowing to solve easily some well-known problems. Thus, in the package "dsl.problems", you can find the class Coloring which helps to solve the problem of coloring a set spaces without having the same color side by side. The Knapsack class has also been added, it models the optimisation problem of filling a bag with maximal utility. Finally, the NQueens class proposes to place N Queens on Chess board where no Queen is able to move to another in a single step, following the chess rules.

1.1 Semantic

Declaration of a variable

Name -> (Range)
Example : "node0" -> (0 to 2) : The variable name will be "node0" and the value that are possible for this variable are 0,1,2.
Shorthand :

- Name : declare a variable of name Name and a range from 0 to 1
- Name to Value : declare a variable of name Name and a range from 0 to Value

Declaration of a constraint

"SumDsl [>==|<==|equal|dif] SumDsl [Sum|RangeVal]"
We also included implicit conversions from IntVar to Sum, from IntVar to SumDSL, from int ro SumDSL, from Sum to SumDSL and from int to Sum which facilitates constraints declarations.

Declaration of a constraint array

"ArrayConstraint [=== | !==] ArrayConstraint"
An ArrayConstraint needs a sum set via a closure.

Sum of variables

"S(Range, Pas default =1, implicit param Name)"
Returns a sum of variables contained in the range incremented by step "Pas". Since each variable must have a unique name, we replace each occurence of % by each value in the range.

Declaration of a problem

First, you need to get the SolverDSL object and initialize it (reset it) to be sure it doesn't contain any old constraint from a previous problem. Then, you can add as many variables and constraints as you want easily, as it has been explained above.

2 Documented Model and Advanced DSL Construct

This part of the report will describe all the classes and objects implemented for this assignment. The DSL code explained is completely contained in the DSL package of the scala project.

2.1 SolverDSL

First, we created a class and companion object called SolverDSL which will be the core of our DSL. It contains a solver and two sets. These sets are respectively used to store all the constraints and variables of a specific problem. In the objective of accessing a variable directly with its name, we also used Map with the variable name as key and this variable as value. The "getItem" method is able to return a specific variable. The parameter of these method is implicit because all variables have the same prefix as name. Thus, using this kind of parameter, a user of our DSL can access a variable by using only its unique number. Defining our SolverDSL as an object, it's easy to access it from other objects. This implementation choice came from the fact that, in RangeVal and Constraint objects, it was really comfortable to add directly constraints and variables to the Solver DSL sets without having to declare an adding method in a class solving a new problem. We rewrote solve and solvewith methods to adapt them to the specificity of our DSL. Indeed, since we don't add directly constraints and variables to the solver, we have to do it before the first call to solve or solveWith. To be sure it's the first call, we decided to use a boolean isAdded to perform the check. We split the process of adding complex constraints (with operators) by using a list. Constraints are added and removed from this list according to constraint construction.

2.2 RangeVal

RangeVal is class extending IntVar which allows to represents a problem variables. The companion object contains several implicit conversions whose a very interesting one starting from String to RangeVal. Since SolerDSL is an object, it's possible to know if that String is a variable name already declared in the DSL and , if appropriate, return this variable;

otherwise, create a new RangeVal with this name.

2.3 Constraint

Constraint is a class representing problem specific constraints. Between them, with our implementation, it's possible to use operators AND (&) and OR (|).

2.4 ArrayConstraint

ArrayConstraint is class which allows to create constraints f First, we created a clasor a set of variables. With this construct, the construction of constraints for a problem can be very scalable. In our tests, it was very useful when two variables couldn't have the same value in a small set of possibilities. Since we don't know in advance the sum number that will compose the ArrayConstraint, we used a closure construct of sums.

2.5 S

The construct S is able to compute the sum of several variables.

2.6 SumDsl

SumDSL object encapsulate Sum and allows us to use all possible operators in order to create constraints. Thus, you can find the following operators : equal, not equal, larger or equal, smaller or equal. We also created a companion object containing several implicit conversions.

3 Conclusion

Our DSL already allows several abstractions and we didn't modified the solver. With our implementation, it's really easy to solve new problems. We facilitated access to variables and constraints, a lot of implicit conversions so a user can add constraints to variables that are not directly linked. Besides problems that are already defined, we added some more test like Sudoku and MagicSquare to convince you that we didn't work in a way to solve only problems given in the assignment. If we had more time, we wanted to add implementation around booleanVar and work even more on helping user to write easily code or sentences that could be understood by the DSL as constraints and variables declaration.