# Domain Specific Language

*INGI 2132 - Final Project*

A SAT solver is a tool to solve instance of the SAT problem (note that the SAT problem was the first known examples of an NP-Complete problem). To do so, an algorithm mixing search and unit-propagation (e.g. DPLL) is applied on the set of clauses to find an assignment such that all the clauses are true.

Unfortunately, reducing problems as instances of the SAT problem is not straightforward. Indeed, it is not always clear how to encode integer problems with arithmetic and high-level constraints in term of literals and clauses. Therefore, using SAT solvers require an important amount of work and high level of expertise on the problem to solve.

The aim of this final project is to develop a Scala DSL on top of Sat4J i.e. a SAT solver implemented in Java. Particularly, your DSL should allow user to model integer satisfaction and optimization problems as naturally as possible. To help you in this task, we provide you with a simple and light solver that already contain the necessary methods to encode integer variables and arithmetic constraints in term of clauses.

## The Solver

The solver is contained in the solver package available on iCampus. The sequel of this section is dedicated to a short introduction to the mechanisms of the solver and its main classes.

`Solver` is a class that provides you with an interface to use Sat4j with integer variables and integer constraints. The following grammar is used to model problems with the solver:

```
Expr    ::= IntVar | Literal
Term    ::= IntVar | Sum((Int, IntVar)*)
Literal ::= BooleanVar | Not(Literal) | LeZero(Sum) |
            And(Literal*) | Or(Literal*)
```

`IntVar` is the class that represents an integer variable. An integer variable has a domain, that is the range of value that could be assigned to the variable. Observe that each variable of the problem need to be added explicitly to the solver using the `addVariable` function.

`Sum` is a class to represent a sum of integers and integer variables. Basically, a sum represents the expression b + a_1 * x_1 + a_2 * x_1 + ... + a_n * x_n where b and a_i are integer (`Int`) and x_i are integer variables (`IntVar`).

`Literal` is an abstract class that represent could be use to model to constraints of the problem. Using the `addConstraint(l: Literal)` a new constraint is declared and added to the solver to enforce the value of the literal `l` to be true. As mentioned above, a problem is satisfiable if it exists a possible assignment of its variables that makes all the literals (added as constraint) true.

`LeZero` is probably the most interesting constraint of the solver since it is the only way to

link integer variables with literals. The literal `LeZero` is true if and only if its inner `Sum` is lower or equal to zero. Using the `Sum` object and the `LeZero` literal, it is possible to define more elaborated constraints. For instance, `x: IntVar < y: IntVar` could be expressed as follows:

```
LeZero(x - y)
```

LeZero can also be combined with other literal to form more elaborate constraints. For instance, the `x: IntVar == y: IntVar` is expressed as follows:

```
And(LeZero(x-y), LeZero(y-x))
```

## Objectives of the project

Here is an non-exhaustive list of criterion that will be used to evaluate your project:

- Is your DSL weakly coupled ? Your DSL should not modify the mechanisms of the solver. Indeed, the DSL has to be developed on top of the solver not inside.
- How easy is it to use the DSL ? How do we declare a new variable/constraint ? Is there any abstraction to model constraints easily ?
- Is your DSL generic ? It should be easy to use your DSL to model new problems that are not contained in the examples of the solver.
- Is your DSL using the concepts presented during the course in an appropriate way ?

## Some references

Some reference for Scala:

- Twitter Scala School: twitter.github.io/scala_school/
- Functional Programming in Scala on Coursera: www.coursera.org/course/progfun
- Scala for the Impatient: available at the BST