Project: Solving proximity constraints

Jan-Michael Holzinger[*] Sophie Hofmanninger[†]

JKU Linz — SS2019

Version Number	Changes Summary	Author
0.1		Jan-Michael
0.2	added System Model	Jan-Michael
0.3	modified Parser, added Workflow	Jan-Michael
0.4	add ConstraintSimplification	Sophie

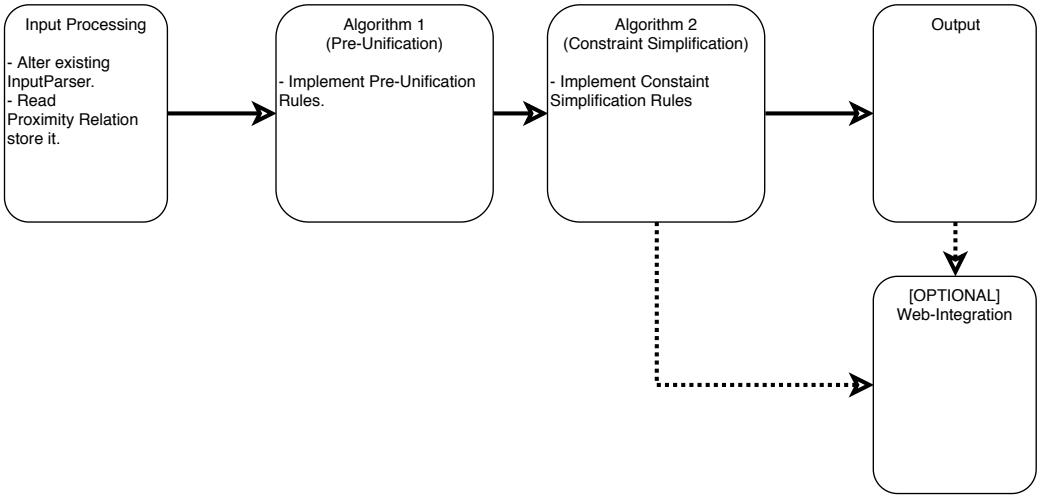
1 System Overview

We split the problem in 4 (5) smaller tasks:

1. Input Processing,
2. Pre-Unification,
3. Constraint Simplification,
4. Output.

O. Web-Integration.

[*]jan.holzinger@gmx.at
[†]sophie@hofmanninger.co.at


Input Processing
- Alter existing InputParser.
- Read Proximity Relation store it.
Algorithm 1
(Pre-Unification)
- Implement Pre-Unification Rules.
Algorithm 2
(Constraint Simplification)
- Implement Constaint Simplification Rules
Output
[OPTIONAL]
Web-Integration

1.1 Input Processing

The first idea here is to copy, alter and extend the existing code, in the class InputParser.

For the Proximity Relations $\mathcal{R}$ and the λ-*cut* we have the following idea:

1. We try to get the number of function symbols (n), constants are treated as 0-ary functions.
2. We let the user input the values to construct a symmetric matrix that consists of values in $[0, 1]$. This matrix must have a 1 in the main diagonal. All values below are 0. Therefore they will not be stored in the implementation.
3. We let the user (later) input $\lambda \in [0, 1]$ and calculate the set $\mathcal{R}_\lambda$.

1.2 Algorithms

We implement the Algorithms in an own class, that has two static functions, preUnification and constraintSimplification.

1.2.1 Pre-Unification Algorithm

The preUnification method consists of a loop, that runs until either $\mathrm{P} = \emptyset$ or it is detected, that there is no solution to the problem.
Inside the loop body, the 7 pre-unification rules are iteratively applied to the first element (which gets popped by doing so).

The method changes the problems constraints and pre-unifier accordingly.

1.2.2 Constraint Simplification Algorithm

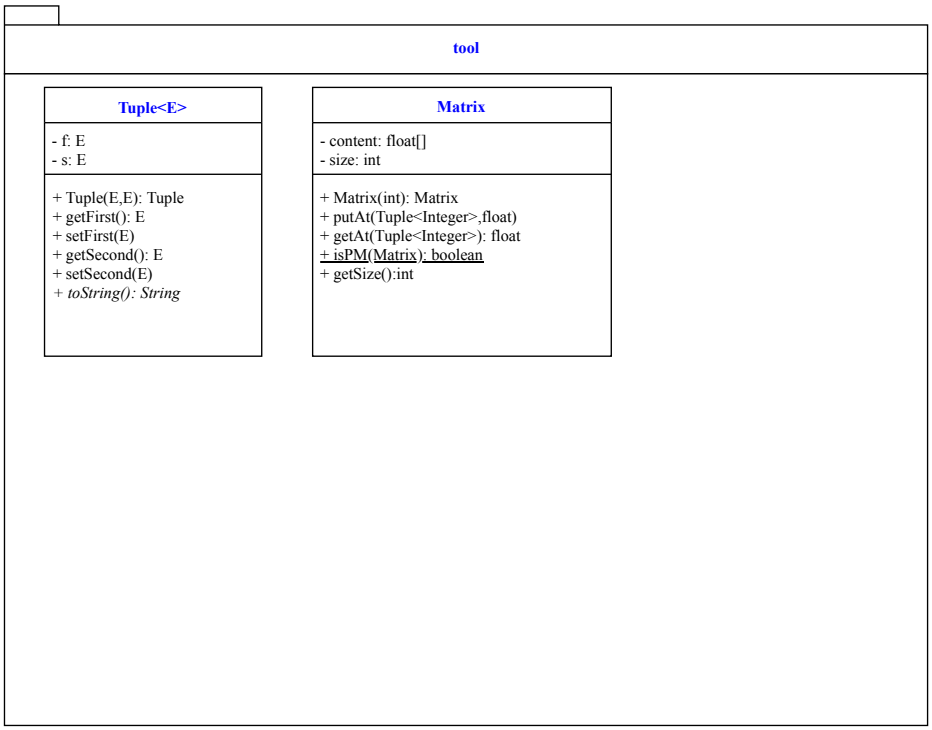
The constraintSimplification method will be called, if the preUnification method returns true. As the preUnification method, the constraintSimplification method consists of a loop. This loop runs until the set of constraints is empty. As input the method needs the problem, more precisely the set of constraints, and the relation matrix $\mathcal{R}$. If the set of constraints could be simplified, the method returns true and otherwise false.
Inside the loop, the seven rules to simplify the constraint set, will be applied. Two of the seven rules are hidden, because we implement the FSN rule directly in the NFS rule and the NN2 rule is integrated in the NN1 rule.

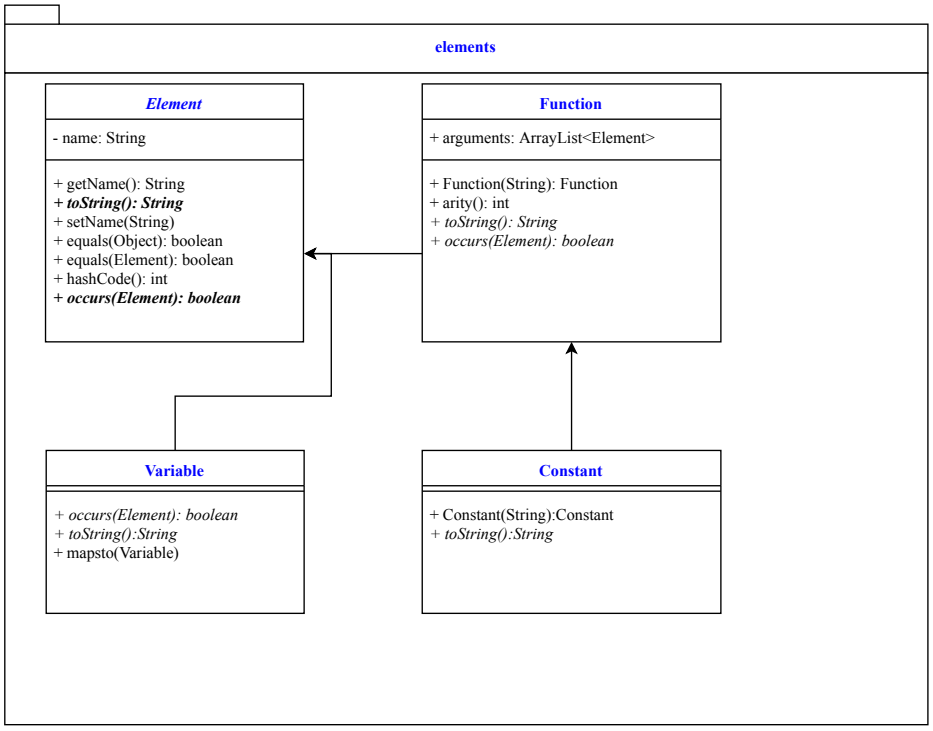
1.3 Output

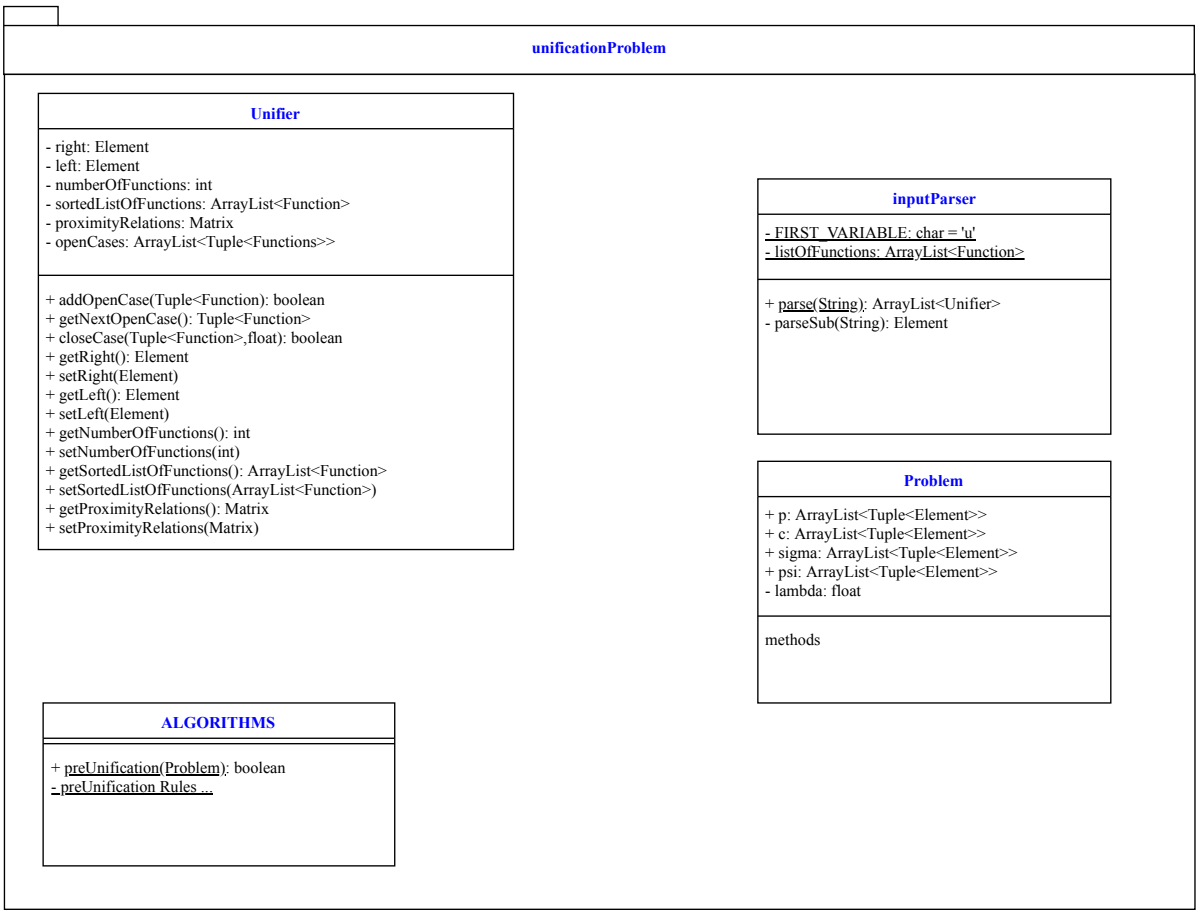
2 System Model

The program consists of 3 packages,

- tool
- elements
- unificationProblem

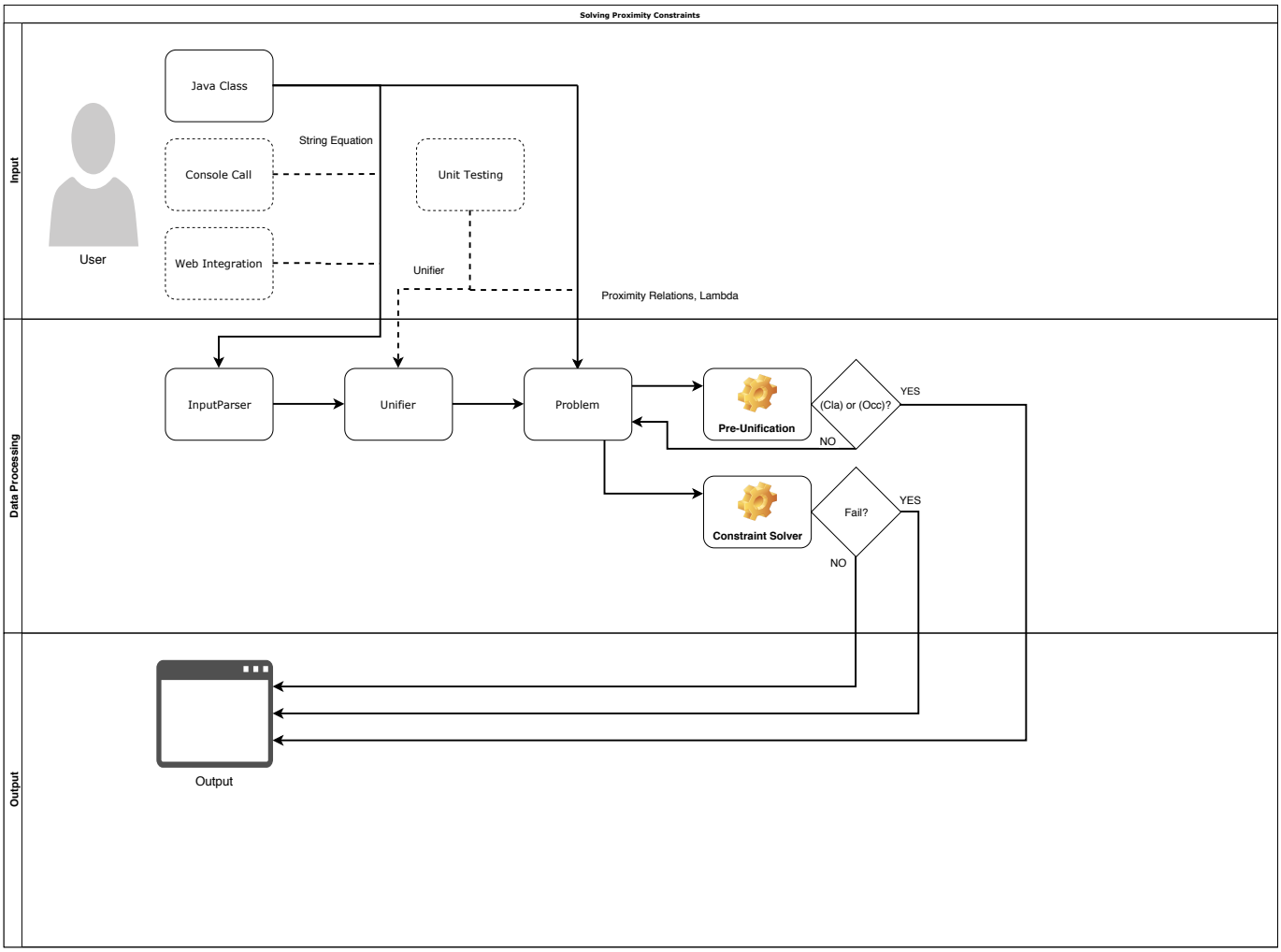

tool
Tuple<E>
- f: E
- s: E
+ Tuple(E,E): Tuple
+ getFirst(): E
+ setFirst(E)
+ getSecond(): E
+ setSecond(E)
+ toString(): String
Matrix
- content: float[]
- size: int
+ Matrix(int): Matrix
+ putAt(Tuple<Integer>,float)
+ getAt(Tuple<Integer>): float
+ isPM(Matrix): boolean
+ getSize():int


elements
Element
- name: String
+ getName(): String
+ toString(): String
+ setName(String)
+ equals(Object): boolean
+ equals(Element): boolean
+ hashCode(): int
+ occurs(Element): boolean
Function
+ arguments: ArrayList<Element>
+ Function(String): Function
+ arity(): int
+ toString(): String
+ occurs(Element): boolean
Variable
+ occurs(Element): boolean
+ toString():String
+ mapsto(Variable)
Constant
+ Constant(String):Constant
+ toString():String


unificationProblem
Unifier
- right: Element
- left: Element
- numberOfFunctions: int
- sortedListOfFunctions: ArrayList<Function>
- proximityRelations: Matrix
- openCases: ArrayList<Tuple<Functions>>
+ addOpenCase(Tuple<Function): boolean
+ getNextOpenCase(): Tuple<Function>
+ closeCase(Tuple<Function>,float): boolean
+ getRight(): Element
+ setRight(Element)
+ getLeft(): Element
+ setLeft(Element)
+ getNumberOfFunctions(): int
+ setNumberOfFunctions(int)
+ getSortedListOfFunctions(): ArrayList<Function>
+ setSortedListOfFunctions(ArrayList<Function>)
+ getProximityRelations(): Matrix
+ setProximityRelations(Matrix)
inputParser
- FIRST_VARIABLE: char = 'u'
- listOfFunctions: ArrayList<Function>
+ parse(String): ArrayList<Unifier>
- parseSub(String): Element
Problem
+ p: ArrayList<Tuple<Element>>
+ c: ArrayList<Tuple<Element>>
+ sigma: ArrayList<Tuple<Element>>
+ psi: ArrayList<Tuple<Element>>
- lambda: float
methods
ALGORITHMS
+ preUnification(Problem): boolean
- preUnification Rules ...

3 Work Flow

The typical workflow looks like this:


Solving Proximity Constraints
Input
Data Processing
Output
Java Class
String Equation
Console Call
Unit Testing
Web Integration
Unifier
User
Proximity Relations, Lambda
InputParser
Unifier
Problem
Pre-Unification
(Cla) or (Occ)?
YES
NO
Constraint Solver
Fail?
YES
NO
Output