CS 512 Formal Methods, Spring 2017 Instructor: Assaf Kfoury

First Order Tableau
Part II

April 13, 2017 Rawane Issa

(These lecture notes are **not** proofread and proof-checked by the instructor.)

Unification

Introduction

For more background on the history of unification, see first page Handout 27 . Note that unification as an area is based on the work of computer scientist. The different variations of unification cover the different complexity classes *(you've probably already at least encountered problems which belong to the class of polynomials: most of the search algorithms you know as a matter of fact!)*. The kind of unification that we need for our purposes, i.e. first order unification, is however very limited compared to other forms of unification that you may find in the literature.

What is the difference between unification and matching?

Unification deals with unifying terms, while matching is the problem of having two terms and applying substitution to only one side in order to match both sides.

Preliminary Definitions

Most of the definitions you will find in this section are presented to you in Handout 27 page 4.

Regarding Substitutions. Before applying the algorithm which will unify your set of terms, you should first note that substitutions are given to you in the form of a mapping from variables to terms. You should however extend this mapping in order to get a mapping from terms to terms.

Definition: Unifier. A unifier is a substitution which, once applied to both sides of the term gives syntactic equality.

Definition: MGU. A most general unifier *(MGU in short)* does the minimal amount of work in order to unify a set of terms.

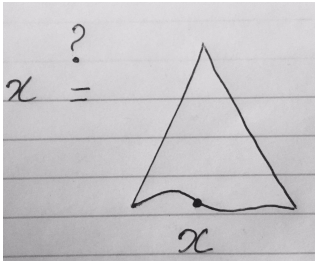
Rules for Unification.

You can find these rule in Handout 27 page 6.

Algorithm Description. The algorithm for unification will work as follows: we will start with a set of instances and we will have to put through a "rewrite system": The terms in this set will be re-written according to a list of rules that will be presented to you in subsequent parts of these notes. Once we are done applying these rules, we will either say that there are no unifiers or, if we end without failure, say that we have a **unifier**.

Unification may be carried out according to 6 unification rules that we will briefly comment on (please refer to Handout 27 page 6. for a more formal definition of each of these rules):

Figure 1: We are trying to unify x with a parse tree containing x as a leaf



1. **Delete rule.** if you have one of the equation such that a terms is "$\stackrel{?}{=}$" to itself, then it's redundant and you should delete it.

2. **Decompose.** if you have a function symbol applied to a set of terms, and the same function symbol applied to other terms, then you can get rid of the function and you unify its arguments.

3. **Conflict** if you have a function symbol applied to a set of terms, and a different function symbol applied to other terms, then you immediately fail and cannot unify.

4. **Orient.** if a terms is "$\stackrel{?}{=}$" a variable, then you can flip the equality and try to unify on the resulting equation.

5. **Eliminate.** In this rule, you keep the equation you are dealing with. However, you then apply a substitution to the rest of x with a non trivial binding which substitutes x with t.

6. **Occurs check.** If you reach a point where you can apply this rules then you automatically get a failure. This rule applies when the equation you want to transform has a variable x on the LHS (left hand side) and a term t on the RHS and $x \in var(t)$ but t is not a variable (its a parse tree): As you can see in Figure 1, This occurs when we are trying to unify x with a parse tree which has x as a leaf. Whatever substitution you invent, you must apply it to both sides of the equation and there is no way for the two sides to unify: you will have to do it again on the leaf x in the parse tree, and both sides will never equalize. There is a kind of a recursive situation that occurs here.

Example.

Suppose we have the following set of equations that we would like to unify:

$$S := \{x \stackrel{?}{=} f(a), g(x,x) \stackrel{?}{=} g(x,y)\} \tag{1}$$

Then by applying the **eliminate** rule we get:

$$\{x \stackrel{?}{=} f(a), g(f(a), f(a)) \stackrel{?}{=} g(f(a), y)\} \tag{2}$$

We then apply a **decompose** rule and get:

$$\{x \stackrel{?}{=} f(a), f(a) \stackrel{?}{=} f(a), f(a) \stackrel{?}{=} y\} \tag{3}$$

Now we apply the **delete** rule in order to get ride of the redundant term:

$$\{x \stackrel{?}{=} f(a), f(a) \stackrel{?}{=} y\} \tag{4}$$

Now **orient** finally gives us:

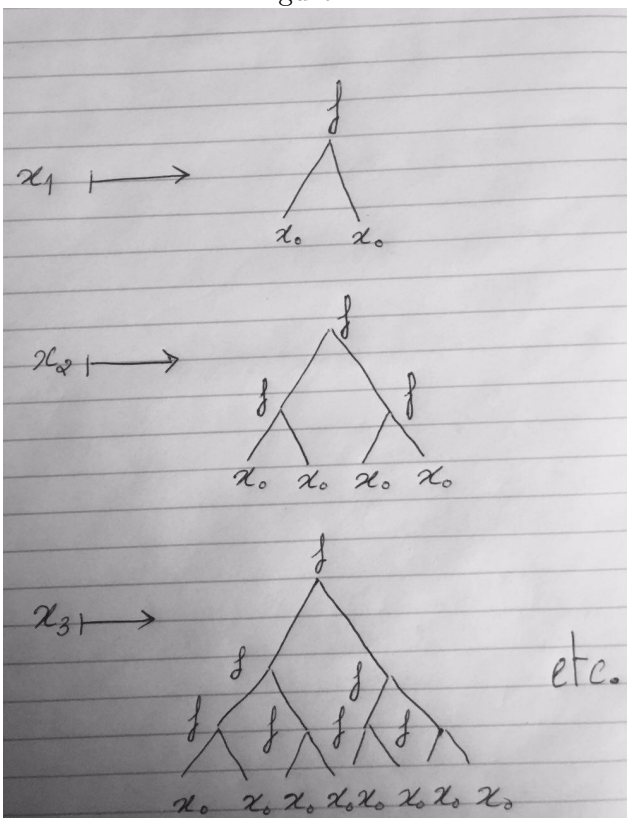
$$\{x \stackrel{?}{=} f(a), y \stackrel{?}{=} f(a)\} \tag{5}$$

We have now arrived at a point where all the previously mentioned rule no longer apply. Since we reached this state without previously reaching any failure then there must be a unifier/solution. The unifier, as mentioned in class, must be "staring us in the face". If look carefully at equation (5) we notice that the solution is actually:

$$\sigma := \{x \mapsto f(a), y \mapsto f(a)\} \tag{6}$$

Time Complexity Considerations

Figure 2:



The algorithm that we have described is an adaptation, or rather a skeleton, of the Martelli Montanari algorithm. As you may have already noticed, the time complexity we achieved in our

adaptation is exponential, while that of the Martelli-Montanari algorithm is polynomial: in order to reach Martelli-Montanari $O(n\log(n))$ time complexity, one would have to cleverly use and exploit a particular kind of data structure that you may have already seen in one of your algorithm courses: DAG's.
In order to clearly see why our algorithm is exponential (and can easily "blow up" in term of time complexity), we will study the following example:

$$S := \{x_1 \stackrel{?}{=} f(x_0, x_0), x_2 \stackrel{?}{=} f(x_1, x_1), x_3 \stackrel{?}{=} f(x_2, x_2), \ldots, x_n \stackrel{?}{=} f(x_{n-1}), f(x_{n-1})\} \tag{7}$$

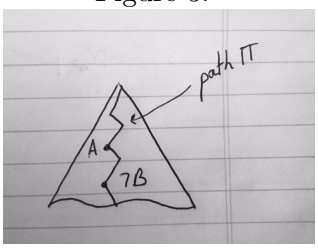
Applying the algorithm:

$$S \stackrel{*}{\to} \{x_1 \stackrel{?}{=} f(x_0, x_0), x_2 \stackrel{?}{=} f(f(x_0, x_0), f(x_0, x_0)), \ldots\} \tag{8}$$

This can be seen as in the form presented in Figure 2.
We can already see that with an input of size $O(n)$, we need $O(2^n)$ steps in order to produce the output. In order to get ride of this exponential blow up then as previously mentioned, rather than using trees, we can use DAGs.

Second Tableau Method: Free variables tableau

Figure 3:



In this section, we will describe how to use unification in order to get the second tableau method that we've mentioned in previous sections.

You can think of any instance of tableau as a tree. Imagine now that in one instance of these instance, there is a path π between literals A and $\neg B$, both of which are atomic. This scenario is drawn for you in Figure 3. A and B could be of the form say $A := Q(f(x, g(x, a)), z)$ (Q is a binary predicate symbol) and $B := Q(y, h(x))$ (where h is a unary function). In such a scenario, we can ask the following question: is there a substitution σ which is the MGU of A and B? If the answer is yes, then close the branch/path π (only this path, not the whole tableau).

We will now provide a brief explanation of the "Rule(σ)" found in Handout 28 page 9: If I have a path π and would like to add a wff φ to it, then we append φ via $T \oplus_\pi \varphi$ (where T is the tableau we have generated so far). This rule extends the tableau. ($X_{\sigma(v_1)}$ means that we closed the path via the substitution σ applied to every v_1 in the tree).

Soundness and Completeness

While we will not discuss this topic in full details, we will mention that the second tableau method is also sound and complete (i.e. the extension of the method to first order does not violate soundness

and completeness). Soundness in this method means that If we can generate a closed tableau from an initial set of sentences in PNF Γ, then Γ is unsatisfiable. Completeness in this method means that if set of sentences in PNF Γ is unsatisfiable, then there exists a closed tableau generated from Γ by these rules.

Comparison between First and Second Tableau methods

You can find an in-depth comparison of the two method Handout 28 pages 12 to 20. Notice that in page 15, applying the ground method (with book keeping notation where for example $5:1$ means that the ID of the node is 5 and this node depends on node 1) will generate around 37 nodes. If we however apply the second method then we gain alot: the tree will contain 5 nodes. We can therefore already tell that there are certain scenarios (similar to the one on page 15) where we can gain in efficiency if we apply one method rather than the other. There are many more strategies and heuristics for using tableau in that you can find in the literature (we have only seen a very small number of them).

That is all what we will mention regarding the topics of unification and tableau.

Introductory notes on Resolution in FOL

You will notice in this section that we have already done much of the work required in order to extend resolution to FOL in the section on resolution in PL: All of the comments found in Handout 11 on resolution for PL will apply for FOL.

This section is based on Handout 29.

Resolution works (page 9) in the following manner: first we transform a wff into CNF, we then skolemize it and get a universal sentence which is equi-satisfiable to our original sentence (Remember that skolemization will produce a sentence of the form $\forall x_1, \ldots, x_k\ \varphi$, where φ is a quantifier free matrix that we will turn into CNF).
We will then have something similarly to the dijunct $C_1 \wedge \ldots \wedge C_n$.

If we compare resolution in PL to resolution in FOL we notice that:

1. In PL C_1 could be of the form $C_1 := \{X \vee Y \vee \neg Z\}$ also written as $C_1 := \{X, Y, \neg Z\}$ for convenience. In FOL C_1 can be of the form $C_1 := \{Q(f(x,z),z) \vee \neg R(f(z))\}$ also written in the form of a set as $C_1 := \{Q(f(x,z),z), \neg R(f(z))\}$.

2. In PL, the resolution rule is applied to two literals which are the complements of one another. In FOL, we will have the notion of a resolvent. however in the case of FOL the literals will be atomic in this case and the resolution pair will be an atomic formula and another atomic formula which are syntactical complements (via some unifier).

3. In FOL we will also have similar situations to PL where depending on the wff, either tableau or resolution will work much better than the other method.

Assume now that we have two clauses C_1 and C_2 in FOL that are given to us in CNF. Also assume that there are literals $P(\vec{s}) \in C_1$ (where $\vec{s} = s_1, \ldots, s_n$)and $\neg P(\vec{t}) \in C_2$ (where $\vec{t} = t_1, \ldots, t_n$). Assume that there is a MGU for these two literals. This means that we can unify their respective

arguments: $MGU(\{s_1 \stackrel{?}{=} t_1, \ldots, s_n \stackrel{?}{=} t_n\})$. Then we can apply the rule:

$$\frac{C_1 \qquad C_2}{(\sigma(C_1) - \{\sigma P(\vec{s})\}) \cup (\sigma(C_2) - \{\sigma \neg P(\vec{t})\})}$$

You might noticed that this idea is a generalization of what we did in PL.