Olivia Lara
Professor Warford
Programming Paradigms
October 29, 2018

Summary and Analysis of "Unification: A Multidisciplinary Survey"

For each section, write a one paragraph description that (a) summarizes in your own words what the section is about, and (b) describes what you consider to be the most important or interesting points. Then, answer the following questions.

The unification problem has been studied in a variety of ways and in a number of fields of computer science. In Kevin Knight's "Unification: A Multidisciplinary Survey," the author discusses the relationships among the many understandings of unification. To begin, the paper defines the *unification problem* as such: a problem that asks if there is a substitution of terms for variables that will make two terms identical. It follows that the paper then defines *substitution* as both a function from variables into terms and a function from terms to terms. It is also important to note that the composition of substitution is associative but not commutative. Moreover, Knight also includes definitions of the following: *variable symbol, constant symbol, function symbol, term, unifiable, unifier, most general unifier, infinitely unifiable,* and *match*. To me, the most important of these terms, however, is *match*, as matching is a key component in unification. Overall, I wasn't aware of the unification problem before reading this article; thus, the fact that there not only exists the problem but also there exists so many relationships among the understandings of the problem is intriguing to me.

The next section reviews the great efforts made to improve the efficiency of unification. Although Herbrand demonstrated a nondeterministic algorithm to compute a unifier of two terms in 1930, the contemporary utilization and notation of unification didn't occur until Robinson introduced a method of theorem proving based on resolution. This is notable because this very method integrated first-order terms. Moreover, other efforts by Guard and Reynolds were made to improve Robinson's original algorithm, but despite their time dedicated to improving the theorem, Robinson' algorithm still proved to be inefficient. Robinson himself devoted more time to improving the efficiency of unification. At this, he realized a shortened representation for terms was necessary and was able to improve the space efficiency of unification. Venturini-Zilli was able to improve the algorithm by introducing a marking scheme in 1975. Additionally, Huet's work on higher order unification resulted in an improved time bound. This improved algorithm by Huet is called an almost *linear algorithm* and is used to test the equivalence of two finite automata. A number of other scholars discovered similar algorithms. In 1982, Martelli and Montanari were able to find an algorithm that-while no longer linear- had a run time of $O(n+m$

log *m)*, where m is the number of distinct variables. Corbin and Bidoit later improve this algorithm. The rest of the section briefly explains efforts made to address the problem of unifying with finite terms. For me, I found it almost inspiring that all these people incorporated other people's ideals and theorems and improved them for everyone. In other words, I find it inspiring they were all so willing to share their knowledge and allow what they found to be further developed or improved. Moreover, I find their acceptance and willingness to spread and further develop ideas to be the key component in this section, as doing this allowed them to get more efficient algorithms and theorems.

The third section begins by briefly explaining a version of Robinson's original unification algorithm. The function UNIFY takes two terms as arguments and returns two items. Here, we are introduced with the *occur check*, which reports true if the first argument occurs anywhere in the second argument. Moreover, Knight notes that Robinson's algorithm didn't contain any flaws in the sense of time or space, but representation of unification. First-order terms can be represented as a linear array, or the *string representation* of a term (Robinson's algorithm utilizes this representation). *Tree representations* are similar to that of *string representations* and they both function effectively when terms are not very complicated. In the case of complicated terms, a *graph representation* is most effective. The algorithms of Heut, Baxter, and Jaffar employ the *graph representation*. Likewise, Paterson and Wegman's linear algorithm and Corbin and Bidoit's algorithm also employ the *graph representation* with an alteration to include *parent pointers*. The last portion of the section talks about a more efficient UNIFY than the one mentioned in the previous one. It differs in that terms are represented as graphs and, so, uses equivalence classes of nodes for merging disjoint sets. Finally, Knight notes that Huet's algorithm is said to be almost linear and that it is nonrecursive, which may increase efficiency. The key component here, I believe, is understanding the difference between *string representation* and *tree representation* from *graph representation* (and *graph representation with pointers*). I took away that string and tree representation function best when terms are not complicated, but in the case that they are it is most effective to use graph representation. Lastly, I find it important to remember from this section that nonrecursive algorithms means increased efficiency, and that Huet's algorithm is more efficient than the previous mens algorithms.

The section opens by explaining that computer science was introduced to the unification problem thanks to Robinson's work on the proving of the resolution theorem. Next, Knight provides an explanation as to how unification is used in resolution theorem proving with an example of two facts that is used to infer a third fact. By changing the three statements into logical form, removing universal quantifiers and implication symbols, we are able to view the statements in clausal form and arrive at the conclusion that resolution is simply the rule of inference that functions to conclude the last clause from the first two. Ultimately, the author explains that the inference rule is so powerful that it is the only rule necessary for a thorough structure of logic.

Naturally, I find the key component of this section to be the power of the inference rule, which is what I think Knight wanted readers to take away from this section.

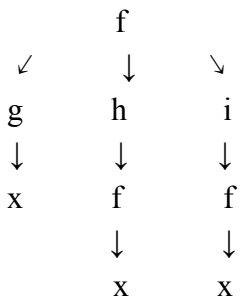
Logic programming was first introduced by Robinson's work. It is important to note that Prolog was the first logic programming language and it is still the most popular logic programming language. Although one might have seen it as merely a forced resolution theorem prover, many men improved it and shaped it into what it is today. Because it utilizes resolution, the programming languages also acquired unification as a key component in the process. And, this utilization of unification is viewed as a "pattern-matching facility to retrieve relevant facts from a database" to Knight. Furthermore, Colmerauer left out the "occur check" as seen in Robinson's implementation of unification. Such a decision allowed for major efficiency gains and caused (and still do cause) Prolog interpreters to leave out the occur check. Knight also notes that like any other search engine, backtracking is required in Prolog when a particular path fails. This causes some of the unifications to be undone in a sense. It is also addressed in the passage that in concurrent programming, unification might have either of the following outcomes: succeed, fail, or suspend. This can prove that it is not possible for unification to be undone. Instead, local copies of structures need to be kept at each unification. The last words of the section explain how various people either presented a new logic language or introduced a variant of Prolog. Nonetheless, I believe it is important to understand that Prolog is a key component to logic programming, as it's the first of its kind and it's still used today. Also, it is important to take away from this passage that the unification in the Concurrent Prolog model doesn't work like the original Prolog- you can't undo unifications.

1. Using Definitions 1.1, 1.2, and 1.3, is $f(x, g(y, b))$ unifiable with $f(f(a),g(g(a, f(a)),b))$?

Yes, they are unifiable.

$$f = f$$
$$x = f(a)$$
$$g = g$$
$$y = g(a, f(a))$$
$$b = b$$

2. Write the tree representation of the term $f(g(x), h(f(x)), i(f(x)))$.



3. Write the graph representation of the term $f(g(x), h(f(x)), i(f(x)))$.