Invariant Generation as Semantic Unification: A New Perspective

Deepak Kapur

Invariant Generation as Semantic Unification: A New Perspective

Deepak Kapur
Department of Computer Science
The University of New Mexico
Albuquerque, NM, USA

—dedicated to my high school friend Pali—

Abstract

Unification is the problem of finding instantiations of variables in a finite set of pairs of terms constructed using function symbols, such that the instantiated terms in each pair become equal. In semantic unification, also called E-unification, function symbols can have properties specified typically by an equational theory; a unifier then makes the instantiated pairs of terms equivalent modulo the equation theory. By generalizing the unification problem further, in which the pairs of formulas have variables ranging over formulas in a first-order theory, the invariant generation problem in software, hardware, and cyber-physical systems can be formulated as a unification problem. Finding a nontrivial unifier then amounts to finding an invariant. Similarly, finding a most general unifier, if unique, amounts to finding the strongest invariant. Instantiation of variables can be further restricted to formulas with certain shapes/properties. This new perspective is illustrated using a number of examples from the literature of the automatic generation of loop invariants in software.

1 Invariant Finding as Unification

The problem of unification in automated reasoning, logic programming, and Artificial Intelligence can be stated as follows: Let S be a finite set of pairs $\{\langle s_i, t_i\rangle \mid 1 \leq i \leq k\}$ of terms s_i, t_i built from a finite set X of variables and a finite set F of constants and function symbols. Check whether there exists a substitution/instantiation σ of variables in s_i, t_i's such that $\sigma(s_i)$ and $\sigma(t_i)$ are identical for every i (referred to as the **unifiability check**), where $\sigma(s_i), \sigma(t_i)$ are the results of applying the instantiation σ on variables in s_i, t_i. If S is unifiable, find a most general unifier (*mgu*) from which all unifiers can be generated. The problem can be further generalized to semantic or E-unification where the semantics of function symbols in F are specified as properties, typically as an equational theory E; in this case, a unifier σ should make $\sigma(s_i) =_E \sigma(t_i)$, where

$=_E$ stands for equivalence modulo E. Higher-order unification extends this problem further to allow function and relation variables to occur in terms that can be appropriately instantiated.

At the University of New Mexico, I have been teaching in my course **Semantics of Programming Languages** for several years that the **program invariant generation** problem can be viewed as a unification problem. In this context, terms s_i's are formulas from a given theory that include formula variables standing for invariants/properties of programs; these formula, variables can be instantiated as formulas.

As illustrated below, a finite set of verification conditions are generated that are expressed using such formula variables, leading to a unification problem.[1] Invariant generation is then to find substitutions for formula variables that make the instantiated verification conditions valid. A more detailed version of this extended abstract/summary is currently being prepared.

Consider a loop **SL** in a computation specified by a program, a hardware or cyber-physical system description, such as:

SL: **while** b do
S
end

A Hoare triple for the above loop is of the form:

$$\{P\} \ \ \mathbf{SL} \ \ \{Q\}.$$

Formulas P and Q are properties of interest for the loop. A Hoare triple specifies that if the program **SL** starts execution in a state satisfying a precondition P, and assuming **SL** halts, the program ends in a state satisfying a postcondition Q.

How does one determine (prove) the validity of the above Hoare triple? An intermediate formula called a *loop invariant,* denoted by I, capturing the partial semantics of the loop body S becomes essential. Using axiomatic semantics, the validity of the above Hoare triple can be decomposed into the following three conditions:

1. $P \Rightarrow I$, 2. $\{(I \wedge b)\} \ \ S \ \ \{I\}$, 3. $(I \wedge \neg b) \Rightarrow Q$.

Conditions 1 and 3 are formulas, called *verification conditions*, while the second condition is still a Hoare triple. The process of associating variables standing for invariants is recursively applied on the loop body S, depending upon its structure, especially when S includes loops. For loop bodies that are sequences of assignment statements, condition 2 can be transformed into a verification condition.

Assuming condition 2 also transforms to one or more verification conditions which are formulas with I (and possibly other formula variables standing for other loop invariants), the above problem is a unification problem. The semantic properties of all symbols other than I and other formula variables, if any, are known. The goal is to find substitutions of I and other formula variables, if any, such that the resulting instantiation of each verification condition is valid. This

[1] Since the second component of pairs in this problem is always **true**, it is omitted.

is analogous to pairs of terms becoming equivalent after instantiation using a unifier.

It should be noted that the above formulation only allows for instantiations of I that are **inductive** invariants of the loop [FK15]; other loop invariants that are not necessarily inductive, can be found by strengthening as discussed in [FK15].

1.1 An Illustrative Example

Consider an example from [Kap06] for computing the floor of the square root of a natural number.

$\{P\}$
$\langle a,\ s,\ t\rangle{:=}\langle 0,\ 1,\ 1\rangle;$
while $s \leq N$ **do**
$\quad \langle a,\ s,\ t\rangle := \langle a+1,\ s+t+2,\ t+2\rangle;$
end while
$\{Q\}$

The axiomatic semantics of a simultaneous multiple assignments statement as a Hoare triple is a generalization of a simple (single) assignment statement.

$$\{P\} \quad \langle x_1, \ldots, x_l\rangle := \langle e_1, \ldots, e_l\rangle \quad \{Q\} \qquad \text{if and only if } \ (P \Rightarrow Q|^{\langle e_1,\ldots,e_l\rangle}_{\langle x_1,\ldots,x_l\rangle}).$$

Here, $Q|^{\langle e_1,\ldots,e_l\rangle}_{\langle x_1,\ldots,x_l\rangle}$ stands for the formula obtained from Q by simultaneously replacing all free occurrences of $x_1, \ldots, x_l$ by $e_1, \ldots, e_l$, respectively. The variables x_i's are assumed to be distinct.[2]

The three conditions discussed above in this case are:

1. $\{P\}\ \langle a,\ s,\ t\rangle{:=}\langle 0,1,1\rangle\ \{I\}$,
which leads to the verification condition: $1'.\ P \Rightarrow I|^{\langle 0,1,1\rangle}_{\langle a,s,t\rangle}$.

2. $\{I \wedge s \leq N\}\ \langle a,\ s,\ t\rangle := \langle a+1,\ s+t+2,\ t+2\rangle\ \{I\}$
from which the verification condition generated using backward semantics is:
$2'.\quad (I \wedge s \leq N) \Rightarrow I|^{\langle a+1,s+t+2,t+2\rangle}_{\langle a,s,t\rangle}$.

As in the case of an expression above, $I|^{\langle a+1,s+t+2,t+2\rangle}_{\langle a,s,t\rangle}$ is the formula obtained from I after simultaneously replacing a, s, t by $a+1, s+t+2, t+2$, respectively.

The third condition is simply $3 : (I \wedge \neg(s \leq N)) \Rightarrow Q$. If Q is known, then it is that specific formula; otherwise, Q is a formula variable. Similarly, if a precondition P is not known, then it is also a formula variable.

In the above example, a goal for the partial correctness of the above program is to find an instantiation $\alpha(a, s, t, N)$ of the loop invariant variable I, a formula

[2] If Q is not quantifier-free, substitutions should be performed carefully to avoid the free variables appearing in e_i's being captured by the scope of quantifiers in Q; to avoid that, quantified variables may need to be renamed. For simplicity, Q is assumed to be quantifier-free.

in input parameters (if any), output parameters (if any), and program variables, as well as for P and Q if they are not known/given, such that these instantiations make the first two verification conditions valid. The instantiation for the post-condition Q can be taken to be any formula implied by $\alpha(a, s, t, N) \wedge \neg(s < N)$, particularly the formula $\alpha(a, s, t, N) \wedge \neg(s < N)$ itself.

It is easy to see that I being **true** satisfies the above conditions. However, this trivial invariant does not provide any useful information about the program's behavior.

Trying to find nontrivial substitutions for I, it can be proved that I being $t = 2a + 1$ and/or $s = -a^2 + at - a + t$ and/or $4s = t^2 - 2a + 3t$ satisfy the above three conditions. From the conjunction of these substitutions, the equation $s = (a + 1)^2$ is implied. It is not a solution to the above unification problem, but its strengthening $s = (a + 1)^2 \wedge t = 2a + 1$ is a solution. Using this substitution, one gets the postcondition $s = (a + 1)^2 \wedge t = 2a + 1 \wedge s > N$, which specifies an approximation to the behavior of the above program.

Observe that a nontrivial solution to the unification problem as formulated above not only gives a loop invariant but also a partial specification of the program. Similarly, a precondition under which a program satisfies its postcondition can be derived from an invariant generated as a unifier.

In the presence of a specific postcondition, the third condition $(I \wedge \neg b) \Rightarrow Q$ constrains possible unifiers. For the above example, a possible postcondition can be $s = (a + 1)^2 \wedge (a + 1)^2 > N \geq a^2$, in which case many of the unifiers for the case where Q is formula variable, are no more any unifiers as they are unable to satisfy the third condition. Notably, **true**, $t = 2a + 1$, or $s = (a + 1)^2$ are no longer unifiers. Any unifier must imply the inequality $N \geq a^2$.

Since a precondition and a postcondition can be arbitrary first-order formulas, the validity problem of first-order logic can be reduced to the above unification problem, showing that the problem is in general undecidable.

2 Constraining Unifiers over a Theory

Let T be a first-order theory associated with a language L consisting of a finite set of predicate and function symbols. Formulas are constructed in the usual way using a finite set X of variables ranging over the domain(s) of the discourse of T. Let FT stand for the formulas in L. To distinguish from the variables in X, include a finite set of new variables ranging over formulas in T. The new variables are called **formula variables** and can be used in the same way as formulas to construct **extended** formulas which include occurrences of formula variables; extended formulas without formula variables are regular formulas. We will abuse the terminology and unless needed, will not make any distinction between extended formulas and formulas.

Given a finite set S of extended formulas $\{\alpha_i | 1 \leq i \leq m\}$ involving formula variables I_j, the unifiability problem is to decide whether there exists a substitution σ for formula variables so that each instantiated formula $\sigma(\alpha_i)$ is valid

in T.[3]

For the above example, formula variables range over the first-order number theory (Peano Arithmetic), which defines the semantics of other symbols such as $0, s, +, *, \geq$. Typically T is desirable to be a decidable theory. If the post-condition in the example is weakened to include only symbols from Presburger arithmetic, then the above unification problem can be solved over that theory. It is also possible to consider an even weaker theory, such as the theory of conjunctions of linear equations over integers. In many such cases when unifiers are restricted to be in a subtheory, the unification problem can be shown to be decidable. This perspective relates to the abstract interpretation framework for generating invariants as discussed in the next subsection; see also section 6.

2.1 Approximation/Abstraction of Verification Conditions

When substitutions for formula variables are restricted to be in a subset of FT, in other words, unifiers are restricted, the verification conditions in S must be approximated to a new problem S' that **soundly approximates** S in the sense that any solution σ of S' is also a solution of S with the extra symbols not in S' being viewed as uninterpreted.

The above example also illustrates the perspective of approximating formulas in S that include occurrences of $<, \geq,$, etc. If the postcondition Q for the above example is a particular formula $s = (a+1)^2$, one can restrict the unification problem in which the formula variables are restricted to be a conjunction of polynomial equalities.

The first condition which includes $\geq$, can be soundly approximated as:
$1'$ $\mathbf{true} \Rightarrow I|^{\langle 0,1,1\rangle}_{\langle a,s,t\rangle}$. The second condition soundly approximates to:

$$2'. \quad I \Rightarrow I|^{\langle a+1, s+t+2, t+2\rangle}_{\langle a,s,t\rangle}.$$

And the third condition is: $3' : \quad I \Rightarrow s = (a+1)^2$.

It can be shown that I can be instantiated as $t = 2a+1 \wedge s = (a+1)^2$, which is thus a unifier.[4] This substitution is also a unifier for the original problem in which the verification conditions are expressed using $<, \geq,$, etc.

Another way to approximate invariant generation as a unification problem is illustrated using an example from a classical paper by Cousot and Halbwachs [CH78] on the polyhedral abstract domain. Conditions/Boolean tests not included in a theory are replaced by formula variables whose substitutions can be restricted to be **true**. The negations of P_1 and P_2 for the loop exit condition and the **if ... then ... else** statement, respectively, are also restricted to be **true** to ensure all paths are considered.

[3] In this formulation of the unifiability problem, the second component in the pair of formulas is always assumed to be **true** and is thus omitted.

[4] It is easy to see that $s = (a+1)^2$ alone is not a unifier since it is not an inductive invariant of the loop; see [FK15].

$\langle i, j\rangle$:=$\langle 2, 0\rangle$;
while P_1 **do**
  if P_2 **then** $\langle i, j\rangle$:=$\langle i + 4, j\rangle$;
  else $\langle i, j\rangle$:=$\langle i + 2, j + 1\rangle$;
  end if
end while

From various program paths, the verification conditions generated leading to a unification problem are as follows:

1. $(i = 2 \wedge j = 0) \Rightarrow I$,
2. $(I \wedge P_1 \wedge P_2) \Rightarrow I|^{\langle i+4,j\rangle}_{\langle i,j\rangle}$ 3. $(I \wedge P_1 \wedge \neg P_2) \Rightarrow I|^{\langle i+2,j+1\rangle}_{\langle i,j\rangle}$.

The postcondition is expressed as a formula variable Q:

4. $(I \wedge \neg P_1) \Rightarrow Q$.

The above unification problem has nontrivial invariants, including $j \geq 0$ as well as $i \geq 2j + 2$. As stated earlier, all unspecified symbols including $P_1, P_2, \neg P_1, \neg P_2$ are assumed to be **true**, representing their weakest possible semantics.

It is also possible to restrict considering substitutions expressed only in certain variables. Such restrictions can be helpful in simplifying unification algorithms.

3 Unification Problems arising from Multiple Loops

As is well-known, for a simple programming language with loops, assignments, conditional statements, and sequencing of statements, it suffices to associate a loop invariant with each loop. Besides formula variables for a precondition and a postcondition, if not specified, the resulting unification problem can have multiple formula variables for each loop.

Consider another related example from [Kap06]:

  $\{P\}$
$\langle m,\ n,\ s,\ x\rangle$:=$\langle 0,\ 0,\ 0, N\rangle$;
while $x \neq 0$ **do**
  $\langle x,\ n,\ m\rangle$:=$\langle x - 1,\ m + 2,\ m + 1\rangle$;
  while $m \neq 0$ **do**
    $\langle s,\ m\rangle$:=$\langle s + 1,\ m - 1\rangle$;
  end while
  m:=n;
end while
  $\{Q\}$

Let I and J, respectively, stand for formula variables associated with the outer loop and inner loop. Corresponding to various paths in the program, the following verification conditions are generated:

$$P \Rightarrow I|^{\langle 0,0,0,N\rangle}_{\langle m,n,s,x\rangle}.$$

$$(I \wedge x \neq 0) \Rightarrow J|^{\langle x-1,m+2,m+1\rangle}_{\langle x,n,m\rangle}$$

for the path from the outer loop entry to the inner loop entry. The path corresponding to the body of the inner loop gives: $(J \wedge m \neq 0) \Rightarrow J|^{\langle s+1,m-1\rangle}_{\langle s,m\rangle}$. And, $(J \wedge m = 0) \Rightarrow I|^{n}_{m}$ is the verification condition corresponding to the path from the exiting of the inner loop to the end of the outer loop body. Finally, upon exiting the outer loop, the verification condition is $(I \wedge x = 0) \Rightarrow Q$.

If I and J are constrained to be conjunctions of linear equalities, then it can be shown that $I: \quad m = 2N - 2x \wedge m = n$ and $J: n = 2N - 2x$ are unifiers. These are indeed invariants of the respective loops under the assumption that P is instantiated to be **true** and Q is I itself.

In Presburger arithmetic, more general unifiers including $m \geq 0,\ n \geq 0,\ s \geq 0,\ x \geq 0$, can also be derived.

If I and J are conjunctions of polynomial equalities and atomic formulas expressing simple inequalities such as $m \geq 0, n \geq 0$, then the above unifiers can be made even more general to include conjunctions of the set of formulas: $\{m \geq 0,\ n \geq 0,\ s \geq 0,\ x \geq 0, m = 2N - 2x,\ m = n,\ \ N^2 - x^2 = s + mx\}$ in I under the assumption that P is instantiated to be $N \geq 0$.

Similarly, J is the conjunction of the set of formulas: $\{\ m \geq 0,\ n \geq 0,\ s \geq 0,\ x \geq 0,\ n = 2N - 2x,\ \ N^2 - x^2 = m + s + n\ x\}$.[5]

4 Invariants/Unifiers form a Lattice with Implication as Ordering

It is easy to prove that if α_1 and α_2 are two unifiers of the verification conditions with formula variables, then $\alpha_1 \wedge \alpha_2$ as and $\alpha_1 \vee \alpha_2$ are also unifiers. Furthermore, if α is a unifier, then so is any formula β such that $\alpha \Rightarrow \beta$. In short, the unifiers of the above problem form a lattice using implication as the ordering. In case a precondition and a postcondition are not specified, i.e., they are also formula variables, then **true** is typically the least element of the lattice, and the strongest unifier corresponding to the strongest invariant is its greatest element.

In the presence of a postcondition, the lattice of unifiers/invariants is constrained by the postcondition; as was illustrated by examples above, some of the formulas that are unifiers/invariants in the absence of a postcondition are not invariants. The lattice of unifiers then has as its least element being the weakest unifier corresponding to the weakest invariant in implication ordering as constrained by the precondition and postcondition. The greatest element still represents the strongest unifier corresponding to the strongest invariant, and need not be constrained by the postcondition.

If unifiers are restricted to be from a subclass of formulas, the lattice of unifiers is the formulas from the subclass with implication relation among them.

[5] Invariants reported of a related program in [Kap06] have quite a few typos, for which I apologize.

5 Unification Algorithms

How does one solve such unification problems? In [Kap06], the quantifier elimination-based approach is discussed in depth for solving some of the above unification problems.

If verification conditions are abstracted to be a conjunction of polynomial equalities and disequalities, implying a conjunction of polynomial equalities, then the unification problem can be transformed into a problem on ideals and their associated varieties. Gröbner basis algorithms can be useful for developing unification algorithms for such cases; see [RC06, RCK04].

If unifiers are restricted to be of a certain shape or have some properties, then a template-based approach presented in [Kap06] can be used for solving the unification problems. This involves hypothesizing a unifier as a template with parameters; from verification conditions to be valid, constraints on parameters are generated. A solution to the parametric constraints after substituting in the template solves the unification problem. Such solutions can be automatically generated using SMT solvers, particularly those solvers which have extensions for synthesizing programs.

In [FK15], an iterative/fixed-point algorithm is given for solving the above problem under the assumption that an *approximate* unifier is already known. A unifier σ is called *approximate* if it does not solve the unification problem but there exists a *strengthening* δ of σ such that $\delta \Rightarrow \sigma$ and δ is a unifier. Typically, δ is obtained by a conjunction of σ with additional atomic formulas generated by the iterative algorithm in its attempt to compute strengthening leading to a unifier.

There are thus many opportunities for developing new unification algorithms when theories are restricted and/or formulas to serve as instantiations for formula variables can only be expressed using certain program variables and/or have a certain shape.

6 Relation to Abstract Interpretation

We conjecture that the above framework encompasses abstract interpretation frameworks, where unifiers/formulas are from an abstract domain such as:

- An **interval** domain in which formula variables are instantiated to be conjunctions of atomic formulas of the forms $x \leq h$ and $l \leq x$, where x is a program variable and l, h are integers or $\pm\infty$, or
- an **octagonal** domain, in which formula variables are instantiated to be conjunctions, including the constraints of the maximum and minimum values on each program variable as well as maximum and minimum values of $x + y, x - y$ for every pair of distinct program variables x, y, or
- a **polyhedral** domain in which formula variables range over a conjunction of linear inequalities.

To solve the verification conditions generated from the semantics of programs, sound approximations may be necessary to get a unifier in the specified domain. Defining an abstraction function and transfer functions on abstract values implementing the operations used in a program is a systematic way to formulate sound approximations to a unification problem such that a solution to the approximate problem is also a solution to the original problem. In [KZH+13], an approach is discussed on how verification conditions can be approximated when invariants are expressed as conjunctions of octagonal constraints. More work is needed to study the relationship between the Abstract Interpretation framework and the proposed formulation of automatic invariant generation.

7 Challenges

There are numerous challenges in viewing an invariant generation problem as a unification problem in theories. Some of these include:

- Developing support for richer data structures such as arrays, pointers, lists, and other containers, as well as other control structures, as long as their axiomatic semantics can be given, leading to the generation of their verification conditions possibly using formula variables.
- Expanding the family of theories on which unification problems formulated for invariant generation can be solved.
- Exploring approximate sound unification problems that lead to generating a unifier for the original problem but are restricted to a formula in a subtheory and/or of a certain shape.

Acknowledgment: Thanks to Jose Abel Castellanos-Joo for his comments while proofreading this extended abstract.

References

[CH78] P. Cousot and N. Halbwachs. Automatic Discovery of Linear Restraints among Variables of a Program. In *Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 84–97, Tucson, Arizona, 1978. ACM Press, New York, NY.

[FK15] S. Falke and D. Kapur. When is a formula a loop invariant? In *Logic, Rewriting and Concurrency: Essays dedicated to Jose Meseguer on the Occasion of His* 65^{th} *Birthday*, LNCS 9200, pages 264–286. Springer-Verlag, 2015.

[Kap06] Deepak Kapur. A quantifier elimination based heuristic for automatically generating inductive assertions for programs. *J. of Systems Sciences and Complexity*, 19(3):307–330, 2006.

[KZH+13] Deepak Kapur, Zhihai Zhang, Matthias Horbach, Hengjun Zhao, Qi Lu, and ThanhVu Nguyen. Geometric quantifier elimination heuristics for automatically generating octagonal and max-plus invariants. In Maria Paola Bonacina and Mark E. Stickel, editors, *Automated Reasoning and Mathematics - Essays in Memory of William W. McCune*, volume 7788 of *Lecture Notes in Computer Science*, pages 189–228. Springer, 2013.

[RC06] E. Rodríguez-Carbonell. *Automatic Generation of Polynomial Invariants for System Verification*. PhD thesis, Universitat Politecnica de Catalunya, 2006.

[RCK04] E. Rodríguez-Carbonell and D. Kapur. Automatic Generation of Polynomial Loop Invariants: Algebraic Foundations. *Intl. Symp. on Symbolic and Algebraic Computation (ISSAC)*, pages 266–273, July 2004.