# EXPLAIN: A Tool for Performing Abductive Inference

Isil Dillig and Thomas Dillig

Computer Science Department, College of William & Mary
{idillig,tdillig}@cs.wm.edu

**Abstract.** This paper describes a tool called EXPLAIN for performing abductive inference. Logical abduction is the problem of finding a simple explanatory hypothesis that explains observed facts. Specifically, given a set of premises $\Gamma$ and a desired conclusion $\phi$, abductive inference finds a simple explanation $\psi$ such that $\Gamma \wedge \psi \models \phi$, and $\psi$ is consistent with known premises $\Gamma$. Abduction has many useful applications in verification, including inference of missing preconditions, error diagnosis, and construction of compositional proofs. This paper gives a brief tutorial introduction to EXPLAIN and describes the basic inference algorithm.

## 1 Introduction

The fundamental ingredient of automated logical reasoning is *deduction*, which allows deriving valid conclusions from a given set of premises. For example, consider the following set of facts:

(1) $\forall x.\ (\text{duck}(x) \Rightarrow \text{quack}(x))$
(2) $\forall x.\ ((\text{duck}(x) \vee \text{goose(x)}) \Rightarrow \text{waddle}(x))$
(3) duck(donald)

Based on these premises, logical deduction allows us to reach the conclusion:

$$\text{waddle(donald)} \wedge \text{quack(donald)}$$

This form of forward deductive reasoning forms the basis of all SAT and SMT solvers as well as first-order theorem provers and verification tools used today.

A complementary form of logical reasoning to deduction is *abduction*, as introduced by Charles Sanders Peirce [1]. Specifically, abduction is a form of backward logical reasoning, which allows inferring likely premises from a given conclusion. Going back to our earlier example, suppose we know premises (1) and (2), and assume that we have observed that the formula $\text{waddle(donald)} \wedge \text{quack(donald)}$ is true. Here, since the given premises do not imply the desired conclusion, we would like to find an explanatory hypothesis $\psi$ such that the following deduction is valid:

$$\frac{\begin{array}{c}\forall x.\ (\text{duck}(x) \Rightarrow \text{quack}(x)) \\ \forall x.\ ((\text{duck}(x) \vee \text{goose(x)}) \Rightarrow \text{waddle}(x)) \\ \psi\end{array}}{\text{waddle(donald)} \wedge \text{quack(donald)}}$$

N. Sharygina and H. Veith (Eds.): CAV 2013, LNCS 8044, pp. 684–689, 2013.

The problem of finding a logical formula $\psi$ for which the above deduction is valid is known as *abductive inference*. For our example, many solutions are possible, including the following:

$$\begin{aligned}
&\psi_1 : \text{duck(donald)} \wedge \neg\text{quack(donald)}\\
&\psi_2 : \text{waddle(donald)} \wedge \text{quack(donald)}\\
&\psi_3 : \text{goose(donald)} \wedge \text{quack(donald)}\\
&\psi_4 \quad \text{duck(donald)}
\end{aligned}$$

While all of these solutions make the deduction valid, some of these solutions are more desirable than others. For example, $\psi_1$ contradicts known facts and is therefore a useless solution. On the other hand, $\psi_2$ simply restates the desired conclusion, and despite making the deduction valid, gets us no closer to explaining the observation. Finally, $\psi_3$ and $\psi_4$ neither contradict the premises nor restate the conclusion, but, intuitively, we prefer $\psi_4$ over $\psi_3$ because it makes fewer assumptions.

At a technical level, given premises $\Gamma$ and desired conclusion $\phi$, abduction is the problem of finding an explanatory hypothesis $\psi$ such that:

$$\begin{aligned}
&(1)\ \Gamma \wedge \psi \models \phi\\
&(2)\ \Gamma \wedge \psi \not\models \text{false}
\end{aligned}$$

Here, the first condition states that $\psi$, together with known premises $\Gamma$, entails the desired conclusion $\phi$. The second condition stipulates that $\psi$ is consistent with known premises. As illustrated by the previous example, there are many solutions to a given abductive inference problem, but the most desirable solutions are usually those that are as simple and as general as possible.

Recently, abductive inference has found many useful applications in verification, including inference of missing function preconditions [2,3], diagnosis of error reports produced by verification tools [4], and for computing underapproximations [5]. Furthermore, abductive inference has also been used for inferring specifications of library functions [6] and for automatically synthesizing circular compositional proofs of program correctness [7].

In this paper, we describe our tool, called EXPLAIN, for performing logical abduction in the combination theory of Presburger arithmetic and propositional logic. The solutions computed by EXPLAIN are both simple and general: EXPLAIN always yields a logically weakest solution containing the fewest possible variables.

## 2 A Tutorial Introduction to EXPLAIN

The EXPLAIN tool is part of the SMT solver MISTRAL, which is available at http://www.cs.wm.edu/~tdillig/mistral under GPL license. MISTRAL is written in C++ and provides a C++ interface for EXPLAIN. In this section, we give a brief tutorial on how to solve abductive inference problems using EXPLAIN.

As an example, consider the abduction problem defined by the premises $x \leq 0$ and $y > 1$ and the desired conclusion $2x - y + 3z \leq 10$ in the theory of linear

```
 1. Term* x = VariableTerm::make("x");
 2. Term* y = VariableTerm::make("y");
 3. Term* z = VariableTerm::make("z");

 4. Constraint c1(x, ConstantTerm::make(0), ATOM_LEQ);
 5. Constraint c2(y, ConstantTerm::make(1), ATOM_GT);
 6. Constraint premises = c1 & c2;

 7. map<Term*, long int> elems;
 8. elems[x] = 2;
 9. elems[y] = -1;
10. elems[z] = 3;
11. Term* t = ArithmeticTerm::make(elems);
12. Constraint conclusion(t, ConstantTerm::make(10), ATOM_LEQ);

13. Constraint explanation = conclusion.abduce(premises);
14. cout << "Explanation: " << explanation << endl;
```

**Fig. 1.** C++ code showing how to use EXPLAIN for performing abduction

integer arithmetic. In other words, we want to find a simple formula $\psi$ such that:

$$x \leq 0 \wedge y > 1 \wedge \psi \models 2x - y + 3z \leq 10$$
$$x \leq 0 \wedge y > 1 \wedge \psi \not\models \mathit{false}$$

Figure 1 shows C++ code for using EXPLAIN to solve the above abductive inference problem. Here, lines 1-12 construct the constraints used in the example, while line 13 invokes the `abduce` method of EXPLAIN for performing abduction. Lines 1-3 construct variables $x, y, z$, and lines 4 and 5 form the constraints $x \leq 0$ and $y > 1$ respectively. In MISTRAL, the operators `&, |, !` are overloaded and are used for conjoining, disjoining, and negating constraints respectively. Therefore, line 6 constructs the premise $x \leq 0 \wedge y > 1$ by conjoining `c1` and `c2`. Lines 7-12 construct the desired conclusion $2x - y + 3z \leq 10$. For this purpose, we first construct the arithmetic term $2x - y + 3z$ (lines 7-11). An `ArithmeticTerm` consists of a map from terms to coefficients; for instance, for the term $2x - y + 3z$, the coefficients of $x, y, z$ are specified as $2, -1, 3$ in the `elems` map respectively.

The more interesting part of Figure 1 is line 13, where we invoke the `abduce` method to compute a solution to our abductive inference problem. For this example, the solution computed by EXPLAIN (and printed out at line 14) is $z \leq 4$. It is easy to confirm that $z \leq 4 \wedge x \leq 0 \wedge y > 1$ logically implies $2x - y + 3z \leq 10$ and that $z \leq 4$ is consistent with our premises.

In general, the abductive solutions computed by EXPLAIN have two theoretical guarantees: First, they contain as few variables as possible. For instance, in our example, although $z - x \leq 4$ is also a valid solution to the abduction problem, EXPLAIN always yields a solution with the fewest number of variables because such solutions are generally simpler and more concise. Second, among the class of solutions that contain the same set of variables, EXPLAIN always yields the *logically weakest* explanation. For instance, in our example, while $z = 0$ is also

a valid solution to the abduction problem, it is logically stronger than $z \leq 4$. Intuitively, logically weak solutions to the abduction problem are preferable because they make fewer assumptions and are therefore more likely to be true.

## 3 Algorithm for Performing Abductive Inference

In this section, we describe the algorithm used in EXPLAIN for performing abductive inference. First, let us observe that the entailment $\Gamma \wedge \psi \models \phi$ can be rewritten as $\psi \models \Gamma \Rightarrow \phi$. Furthermore, in addition to entailing $\Gamma \Rightarrow \phi$, we want $\psi$ to obey the following three requirements:

1. The solution $\psi$ should be consistent with $\Gamma$ because an explanation that contradicts known premises is not useful
2. To ensure the simplicity of the explanation, $\psi$ should contain as few variables as possible
3. To capture the generality of the abductive explanation, $\psi$ should be no stronger than any other solution $\psi'$ satisfying the first two requirements

Now, consider a *minimum satisfying assignment* (MSA) of $\Gamma \Rightarrow \phi$. An MSA of a formula $\varphi$ is a partial satisfying assignment of $\varphi$ that contains as few variables as possible. The formal definition of MSAs as well as an algorithm for computing them are given in [8]. Clearly, an MSA $\sigma$ of $\Gamma \Rightarrow \phi$ entails $\Gamma \Rightarrow \phi$ and satisfies condition (2). Unfortunately, an MSA of $\Gamma \Rightarrow \phi$ does not satisfy condition (3), as it is a logically strongest solution containing a given set of variables.

Given an MSA of $\Gamma \Rightarrow \phi$ containing variables $V$, we observe that a logically weakest solution containing only $V$ is equivalent to $\forall \overline{V}.\ (\Gamma \Rightarrow \phi)$, where $\overline{V} = \text{free}(\Gamma \Rightarrow \phi) - V$. Hence, given an MSA of $\Gamma \Rightarrow \phi$ consistent with $\Gamma$, an abductive solution satisfying all conditions (1)-(3) can be obtained by applying quantifier elimination to $\forall \overline{V}.\ (\Gamma \Rightarrow \phi)$.

Thus, to solve the abduction problem, what we want is a largest set of variables $X$ such that $(\forall X.(\Gamma \Rightarrow \phi)) \wedge \Gamma$ is satisfiable. We call such a set of variables $X$ a *maximum universal subset* (MUS) of $\Gamma \Rightarrow \phi$ with respect to $\Gamma$. Given an MUS $X$ of $\Gamma \Rightarrow \phi$ with respect to $\Gamma$, the desired solution to the abductive inference problem is obtained by eliminating quantifiers from $\forall X.(\Gamma \Rightarrow \phi)$ and then simplifying the resulting formula with respect to $\Gamma$ using the algorithm from [9].

Pseudo-code for our algorithm for solving an abductive inference problem defined by premises $\Gamma$ and conclusion $\phi$ is shown in Figure 2. The `abduce` function given in lines 1-5 first computes an MUS of $\Gamma \Rightarrow \phi$ with respect to $\Gamma$ using the helper `find_mus` function. Given such a maximum universal subset $X$, we obtain a quantifier-free abductive solution $\chi$ by applying quantifier elimination to the formula $\forall X.(\Gamma \Rightarrow \phi)$. Finally, at line 4, to ensure that the final abductive solution does not contain redundant subparts that are implied by the premises, we apply the simplification algorithm from [9] to $\chi$. This yields our final abductive solution $\psi$ which satisfies our criteria of minimality and generality and that is not redundant with respect to the original premises.

```
abduce(φ, Γ) {
1.  φ = (Γ ⇒ φ)
2.  Set X = find_mus(φ, Γ, free(φ), 0)
3.  χ = elim(∀X.φ)
4.  ψ = simplify(χ, Γ)
5.  return ψ
}

find_mus(φ, Γ, V, L) {
6.  If V = ∅ or |V| ≤ L  return ∅
7.  U = free(φ) − V
8.  if( UNSAT (Γ ∧ ∀U.φ))  return ∅

9.  Set best = ∅
10.  choose x ∈ V

11.  if(SAT(∀x.φ)) {
12.   Set Y = find_mus(∀x.φ, Γ, V \ {x}, L − 1);
13.   If (|Y| + 1 > L) { best = Y ∪ {x}; L = |Y| + 1 }
     }
14.  Set Y = find_mus(φ, Γ, V \ {x},L);
15. If (|Y| > L) { best = Y }

16. return best;
}
```

**Fig. 2.** Algorithm for performing abduction

The function `find_mus` used in `abduce` is shown in lines 6-16 of Figure 2. This algorithm directly extends the `find_mus` algorithm we presented earlier in [8] to exclude universal subsets that contradict $\Gamma$. At every recursive invocation, `find_mus` picks a variable $x$ from the set of free variables in $\varphi$. It then recursively invokes `find_mus` to compute the sizes of the universal subsets with and without $x$ and returns the larger universal subset. In this algorithm, $L$ is a lower bound on the size of the MUS and is used to terminate search branches that cannot improve upon an existing solution. Therefore, the search for an MUS terminates if we either cannot improve upon an existing solution $L$, or the universal subset $U$ at line 7 is no longer consistent with $\Gamma$. The return value of `find_mus` is therefore a largest set $X$ of variables for which $\Gamma \wedge \forall X.\varphi$ is satisfiable.

## 4 Experimental Evaluation

To explore the size of abductive solutions and the cost of computing such solutions in practice, we collected 1455 abduction problems generated by the Compass program analysis system for inferring missing preconditions of functions. In each abduction problem $(\Gamma \wedge \psi) \Rightarrow \phi$, $\Gamma$ represents known invariants, and $\phi$ is the weakest precondition of an assertion in some function $f$. Hence, the

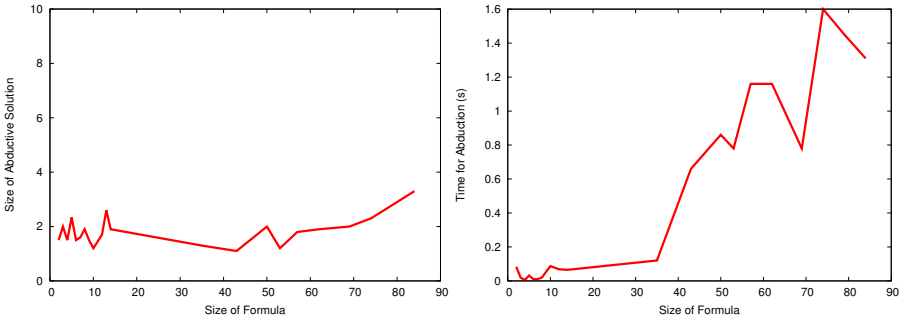


**Fig. 3.** Size of Formula vs. Size of Abductive Solution and Time for Abduction

solution $\psi$ to the abduction problem represents a potential missing precondition of $f$ sufficient to guarantee the safety of the assertion.

The left-hand side of Figure 3 plots the size of the formula $\Gamma \Rightarrow \phi$, measured as the number of leaves in the formula, versus the size of the computed abductive solution. As this graph shows, the abductive solution is generally much smaller than the original formula, demonstrating that our abduction algorithm generates small explanations in practice. The right-hand side of Figure 3 plots the size of the formula $\Gamma \Rightarrow \phi$ versus the time taken to solve the abduction problem. As expected, the time increases with formula size, but remains tractable even for the largest abduction problems in our benchmark set.

## References

1. Peirce, C.: Collected papers of Charles Sanders Peirce. Belknap Press (1932)
2. Calcagno, C., Distefano, D., O'Hearn, P., Yang, H.: Compositional shape analysis by means of bi-abduction. POPL 44(1), 289–300 (2009)
3. Giacobazzi, R.: Abductive analysis of modular logic programs. In: Proceedings of the 1994 International Symposium on Logic Programming, Citeseer, pp. 377–391 (1994)
4. Dillig, I., Dillig, T., Aiken, A.: Automated error diagnosis using abductive inference. In: PLDI (2012)
5. Gulwani, S., McCloskey, B., Tiwari, A.: Lifting abstract interpreters to quantified logical domains. In: POPL, pp. 235–246. ACM (2008)
6. Zhu, H., Dillig, I., Dillig, T.: Abduction-based inference of library specifications for source-sink property verification. In: Technical Report, College of William & Mary (2012)
7. Li, B., Dillig, I., Dillig, T., McMillan, K., Sagiv, M.: Synthesis of circular compositional program proofs via abduction. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 370–384. Springer, Heidelberg (2013)
8. Dillig, I., Dillig, T., McMillan, K.L., Aiken, A.: Minimum satisfying assignments for SMT. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 394–409. Springer, Heidelberg (2012)
9. Dillig, I., Dillig, T., Aiken, A.: Small formulas for large programs: On-line constraint simplification in scalable static analysis. In: Cousot, R., Martel, M. (eds.) SAS 2010. LNCS, vol. 6337, pp. 236–252. Springer, Heidelberg (2010)