

# Weakening Goals in Logical Specifications

Ben M. Andrew 

University of Manchester, Manchester, UK  
[benjamin.andrew@manchester.ac.uk](mailto:benjamin.andrew@manchester.ac.uk)

**Abstract.** Logical specifications are widely used to represent software systems and their desired properties. Under system degradation or environmental changes, commonly seen in complex real-world robotic systems, these properties may no longer hold and so traditional verification methods will simply fail to construct a proof. However, weaker versions of these properties do still hold and can be useful for understanding the system’s behaviour in uncertain conditions, as well as aiding compositional verification. We present a counterexample-guided technique for iteratively weakening properties, apply it to propositional logic specifications, and discuss planned extensions to state-based representations.

## 1 Introduction

Software systems, along with properties that we are interested in proving about them, are often specified in logics such as first-order or temporal logic. Many verification techniques exist for automatically checking whether desired properties hold in a system, but in complex systems that interact with the real world, unexpected environmental conditions or system degradation can cause these properties to no longer hold [9].

In these cases traditional formal verification techniques will simply report property violations, leaving us unable to say anything about the system’s behaviour. However, for many applications we may be interested in weakened forms of the property that do hold in the system.

As an example, consider a quadrotor drone that we have proven can only safely land when the wind speed is below a certain threshold, and now imagine that this property does *not* hold when one of the rotors has failed. There may still be a weaker version of this property, for example with a lower threshold on the wind speed, that does hold for the degraded drone. Being able to automatically deduce this weakened property may be crucial for regulatory approval or understanding how the system properties change under uncertain conditions.

To be precise, a weakening of a property  $P$  is any property that specifies a superset of the behaviours of  $P$ . For example, we can logically specify the above example as weakening  $\text{HighWind} \vee \text{LowWind} \rightarrow \text{CanLand}$  to  $\text{LowWind} \rightarrow \text{CanLand}$ . (Note that strengthening the antecedent weakens the implication.)

Automatic deduction of weakened properties is also useful in compositional assume-guarantee reasoning [8], where our system is an individual component

providing guarantees that feed into the assumptions of other components' specifications [11], and aid proofs of the composed system's global properties.

This PhD project currently aims to answer two core research questions:

**RQ1:** How can a system property, normally holding but invalidated by system degradation or environmental changes, be automatically weakened so that it both holds in the degraded system and is still useful?

**RQ2:** In compositional assume-guarantee reasoning, how does the weakening of a component's guaranteed properties affect other components or system-level properties?

*Related Work.* Goal weakening has been explored in requirements engineering [12]. However, conflicts are only handled between goals because the weakening is done at requirements engineering-time, whereas our approach is concerned with inconsistencies between the requirements and the implementation, which occur at a later stage in the development lifecycle.

Belief revision [6] is a technique where a belief set is updated when new information conflicts with existing beliefs, removing those that conflict. However, this is a coarse approach that does not weaken the individual beliefs themselves, and so can be overly conservative.

Counterexample-guided techniques have been applied to areas like abstraction refinement [3], inductive synthesis [1], and control [7]. However, they have not yet been applied to the problem of weakening goals in specifications.

## 2 Counterexample-Guided Weakening

Our approach finds counterexamples that show that the property doesn't hold in the system, integrates them into the property, and repeats. By integrating counterexamples we iteratively weaken the property until it holds in the system.

Our algorithm, initially applied to propositional logic, is implemented in OCaml, using the Why3 [5] platform with Alt-Ergo [4], a tableau-based solver. The code is hosted publicly on GitHub<sup>1</sup>.

Not all weakenings of the desired property are useful, as evidenced by the trivial property  $\perp$  that any system guarantees. Thus, along with our desired property we also specify a *critical* property  $P_C$  that our system must satisfy, as the minimum weakening of  $P_D$  that we allow.

Our specifications are triples  $\langle A, P_D, P_C \rangle$  of propositional formulae, where  $A$  represents the internal structure of the system and the environment,  $P_D$  represents the desired property of the system, and  $P_C$  represents the critical property of the system.  $\langle A, P_D, P_C \rangle$  is well-formed if and only if  $P_D$  implies  $P_C$ , i.e.  $P_D \rightarrow P_C$ . We begin the proof process by checking whether the iterative algorithm is necessary:

1. Check that  $A \rightarrow P_D$ . If true, then finish successfully with  $P_D$  as the property; otherwise,

---

<sup>1</sup> <https://github.com/benmandrew/prop-goal-weakening>

2. Check that  $A \rightarrow P_C$ . If false, then finish unsuccessfully, as our critical property does not hold; otherwise,
3. Find an intermediate property  $P_I$  between  $P_D$  and  $P_C$  such that  $A \rightarrow P_I$ .

(By  $P_I$  being *between*  $P_D$  and  $P_C$ , we mean that  $P_D \rightarrow P_I$  and  $P_I \rightarrow P_C$ , considering propositional formulae to be partially ordered by implication.)

*Algorithm.* The algorithm uses a counterexample-guided approach, iteratively computing counterexamples using a SAT solver and integrating them back into the candidate property until it is satisfied. It is detailed below as well as in Fig. 1.

The  $i$ -th candidate property is denoted by  $P_I^i$ , for  $i \in \mathbb{N}$ , and we begin by initialising  $P_I^0 = P_D$ . We then construct a formula  $F(i)$  that is a conjunction of the following:

1.  $A \rightarrow P_I^i$ , the candidate property must hold in the system,
2.  $P_D \rightarrow P_I^i$ , the candidate property must be weaker than or equivalent to the desired property, and
3.  $P_I^i \rightarrow P_C$ , the candidate property must be stronger than or equivalent to the critical property.

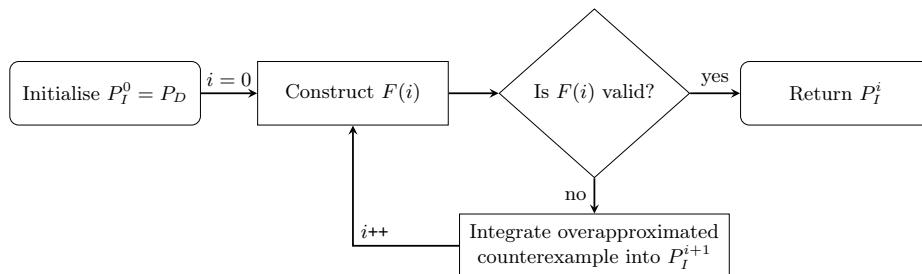
We check the validity of  $F(i)$  using a SAT solver. If  $F(i)$  is valid, then  $P_I^i$  holds in the system and we are finished. Otherwise, the SAT solver gives us a counterexample model that holds in  $A$  but does not in  $P_I^i$ . This model may contain assignments to *hidden* variables that occur in  $A$  but not in  $P_D$  or  $P_C$ . Including them would expose the inner logical workings of the system which may sometimes be desirable, but for the purposes of overapproximating the weakened property, we remove all hidden variables from the counterexample.

From this overapproximated counterexample we construct a formula  $C_i$  which is a conjunction of positive or negative propositional variables. For example, the model  $\{X = \text{true}, Y = \text{false}, Z = \text{false}\}$  corresponds to the formula  $X \wedge \neg Y \wedge \neg Z$ .

The next iteration of the candidate property is then

$$P_I^{i+1} = P_I^i \vee C_i$$

and we repeat the loop by constructing and checking  $F(i + 1)$ .



**Fig. 1.** Overview of the algorithm, which iteratively weakens the candidate property  $P_I^i$  until it is satisfied by the system representation  $A$ .

The algorithm is complete for systems with finite numbers of variables. As each step adds at least one complete interpretation to the candidate property, the number of iterations is bounded by  $2^N$ , where  $N$  is the number of unique propositional variables in  $P_D$  and  $P_C$  combined.

*Example.* We use the example from the introduction concerning a quadrotor drone. Our propositional variables are  $R_4$ , that all four rotors work;  $R_3$ , that only three rotors work;  $W_H$ , that windspeed is high;  $W_L$ , that windspeed is low; and  $L$ , that the drone can land.

The system is modelled by three assumptions,

$$A = \underbrace{R_3}_{(3a)} \wedge \underbrace{(R_4 \wedge (W_H \vee W_L) \rightarrow L)}_{(3b)} \wedge \underbrace{(R_3 \wedge W_L \rightarrow L)}_{(3c)}$$

which specify (3a), that only three rotors work (i.e. one rotor has failed); and (3b, 3c), the conditions for the drone being able to land. Our desired goal property is  $P_D = (W_H \vee W_L) \rightarrow L$ , but this is not satisfied by the assumptions, so we must weaken it. (For the purposes of demonstration we let our critical property  $P_C = \top$ .)

We construct the initial  $F(0)$  with  $P_I^0 = P_D$ , and check for validity with the SAT solver, receiving a negative answer with the counterexample  $\neg L \wedge W_H \wedge \neg W_L \wedge R_3 \wedge \neg R_4$ . This counterexample contains the ‘hidden’ variables  $R_3$  and  $R_4$ , and as we would prefer not to expose the inner state of our system, we remove them, resulting in the overapproximated counterexample  $\neg L \wedge W_H \wedge \neg W_L$ . Integrating this into our candidate property results in

$$\begin{aligned} P_I^1 &= ((W_H \vee W_L) \rightarrow L) \vee (\neg L \wedge W_H \wedge \neg W_L) \\ &= W_L \rightarrow L \end{aligned}$$

Which is a valid property of the system and so we are done.

### 3 Future Work

To answer **RQ1**, we are investigating how to extend weakening to properties expressed in state-based specification languages, such as Deterministic Finite Automata (DFAs), Buchi automata (which commonly correspond with LTL formulae), and Abstract State Machines (ASMs). We are currently exploring how these properties can be automatically weakened, based on the framework of automata learning with the  $L^*$  algorithm [2].

Weakening goals is not the only way to weaken a specification: in contract-based reasoning [11], strengthening the corresponding assumption serves the same purpose, and may be a more natural solution for changes in the environment. This will contribute to answering **RQ2**. It remains to be seen when this would be appropriate, and how exactly it would be done.

It may be more suitable to frame weakening as an interpolation problem [10] — that is, finding a suitable interpolant between the desired and critical properties, subject to the constraint of being a valid property of the system. This approach requires investigation.

## References

- [1] R. Alur, R. Bodik, G. Jiniwal, M.M.K. Martin, M. Raghothaman, S.A. Seshia, R. Singh, A. Solar-Lezama, E. Torlak, and A. Udupa. “Syntax-Guided Synthesis”. In: *Formal Methods in Computer-Aided Design*. IEEE, 2013, pp. 1–8. DOI: [10.1109/FMCAD.2013.6679385](https://doi.org/10.1109/FMCAD.2013.6679385).
- [2] D. Angluin. “Learning Regular Sets from Queries and Counterexamples”. In: *Information and Computation* 75.2 (1987), pp. 87–106. DOI: [10.1016/0890-5401\(87\)90052-6](https://doi.org/10.1016/0890-5401(87)90052-6).
- [3] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. “Counterexample-Guided Abstraction Refinement”. In: *Computer Aided Verification*. Springer, 2000, pp. 154–169. DOI: [10.1007/10722167\\_15](https://doi.org/10.1007/10722167_15).
- [4] S. Conchon, A. Coquereau, M. Iguernlala, and A. Mebsout. “Alt-Ergo 2.2”. In: *International Workshop on Satisfiability Modulo Theories*. 2018. URL: <https://inria.hal.science/hal-01960203>.
- [5] J. Filliâtre and A. Paskevich. “Why3 — Where Programs Meet Provers”. In: *Programming Languages and Systems*. Vol. 7792. LNCS. Springer, 2013, pp. 125–128. DOI: [10.1007/978-3-642-37036-6\\_8](https://doi.org/10.1007/978-3-642-37036-6_8).
- [6] P. Gärdenfors. *Belief Revision*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1992. DOI: [10.1017/CBO9780511526664](https://doi.org/10.1017/CBO9780511526664).
- [7] T.A. Henzinger, R. Jhala, and R. Majumdar. “Counterexample-Guided Control”. In: *Automata, Languages and Programming*. Springer, 2003, pp. 886–902. DOI: [10.1007/3-540-45061-0\\_69](https://doi.org/10.1007/3-540-45061-0_69).
- [8] C.B. Jones. “Tentative Steps Toward a Development Method for Interfering Programs”. In: *ACM Transactions on Programming Languages and Systems* 5.4 (1983), pp. 596–619. DOI: [10.1145/69575.69577](https://doi.org/10.1145/69575.69577).
- [9] M. Luckcuck, M. Farrell, L.A. Dennis, C. Dixon, and M. Fisher. “Formal Specification and Verification of Autonomous Robotic Systems: A Survey”. In: *ACM Computing Surveys* 52.5 (2019), 100:1–100:41. DOI: [10.1145/3342355](https://doi.org/10.1145/3342355).
- [10] K.L. McMillan. “Applications of Craig Interpolants in Model Checking”. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 3440. Springer, 2005, pp. 1–12. DOI: [10.1007/978-3-540-31980-1\\_1](https://doi.org/10.1007/978-3-540-31980-1_1).
- [11] B. Meyer. “Applying ‘Design by Contract’”. In: *Computer* 25.10 (1992), pp. 40–51. DOI: [10.1109/2.161279](https://doi.org/10.1109/2.161279).
- [12] A. van Lamsweerde and E. Letier. “Handling Obstacles in Goal-Oriented Requirements Engineering”. In: *IEEE Transactions on Software Engineering* 26.10 (2000), pp. 978–1005. DOI: [10.1109/32.879820](https://doi.org/10.1109/32.879820).