Environment Assumptions for Synthesis

Krishnendu Chatterjee\(^2\), Thomas A. Henzinger\(^1\), and Barbara Jobstmann\(^1\)

\(^1\) EPFL, Lausanne
\(^2\) University of California, Santa Cruz

Abstract. The synthesis problem asks to construct a reactive finite-state system from an \(\omega\)-regular specification. Initial specifications are often unrealizable, which means that there is no system that implements the specification. A common reason for unrealizability is that assumptions on the environment of the system are incomplete. We study the problem of correcting an unrealizable specification \(\phi\) by computing an environment assumption \(\psi\) such that the new specification \(\psi \rightarrow \phi\) is realizable. Our aim is to construct an assumption \(\psi\) that constrains only the environment and is as weak as possible. We present a two-step algorithm for computing assumptions. The algorithm operates on the game graph that is used to answer the realizability question. First, we compute a safety assumption that removes a minimal set of environment edges from the graph. Second, we compute a liveness assumption that puts fairness conditions on some of the remaining environment edges. We show that the problem of finding a minimal set of fair edges is computationally hard, and we use probabilistic games to compute a locally minimal fairness assumption.

1 Introduction

Model checking has become a successful verification technique in hardware and software design. Although the method is automated, the success of a verification process highly depends on the quality of the specification. Writing correct and complete specifications is a tedious task: it usually requires several iterations until a satisfactory specification is obtained. Specifications are often too weak (e.g., they may be vacuously satisfied \([2,14]\)); or too strong (e.g., they may allow too many environment behaviors), resulting in spurious counterexamples. In this work we automatically strengthen the environment constraints within specifications whose assumptions about the environment behavior are so weak as to make it impossible for a system to satisfy the specification.

Automatically deriving environment assumptions has been studied from several points of view. For instance, in circuit design one is interested in automatically constructing environment models that can be used in test-bench generation \([21,19]\). In compositional verification, environment assumptions have been generated as the weakest input conditions under which a given software or hardware component satisfies a given specification \([4,6]\). We follow a different path by leaving the design out of the picture and deriving environment assumptions from the specification alone. Given a specification, we aim to compute a least restrictive environment that allows for an implementation of the specification. The assumptions that we compute can assist the designer in different ways. They can be used as baseline necessary conditions in component-based model checking. They can be used in designing interfaces and generating test cases.
for components before the components themselves are implemented. They can provide insights into the given specification. And above all, in the process of automatically constructing an implementation for the given specification ("synthesis"), they can be used to correct the specification in a way that makes implementation possible.

While specifications of closed systems can be implemented if they are satisfiable, specifications of open systems can be implemented if they are realizable —i.e., there is a system that satisfies the specification without constraining the inputs. The key idea of our approach is that given a specification \( \phi \), if \( \phi \) is not realizable, it cannot be complete and has to be weakened by introducing assumptions on the environment of the system. We do this by computing an assumption \( \psi \) such that the new specification \( \psi \Rightarrow \phi \) is realizable. Our aim is to construct a condition \( \psi \) that does not constrain the system and is as weak as possible. The notion that \( \psi \) must constrain only the environment can be captured by requiring that \( \psi \) itself is realizable for the environment —i.e., there exists an environment that satisfies \( \psi \) without constraining the outputs of the system. The notion that \( \psi \) be as weak as possible is more difficult to capture. We will show that in certain situations, there is no unique weakest environment-realizable assumption \( \psi \), and in other situations, it is NP-hard to compute such an assumption.

Example. During a typical effort of formally specifying hardware designs \cite{5}, some specifications were initially not realizable. One specification that was particularly difficult to analyze can be simplified to the following example. Consider a system with two input signals \( r \) and \( c \), and one output signal \( g \). The specification requires that (i) every request is eventually granted starting from the next time step, written in linear temporal logic as \( \square(r \rightarrow \bigcirc\Diamond g) \); and (ii) whenever \( c \) or \( g \) are high, then \( g \) has to stay low in the next time step, written \( \square((c \lor g) \rightarrow \bigcirc \neg g) \). This specification is not realizable because the environment can force, by sending \( c \) all the time, that \( g \) has to stay low forever (Part (ii)). Thus requests cannot be answered, and Part (i) is violated.

One assumption that makes this specification realizable is \( \psi_1 = \square\neg c \). This assumption is undesirable because it forbids the environment to send \( c \). A system synthesized with this assumption would ignore the signal \( c \). Assumptions \( \psi_2 = \square\Diamond\neg c \) and \( \psi_3 = \square(r \rightarrow \Diamond\neg c) \) are more desirable but still not satisfactory: \( \psi_2 \) forces the environment to lower \( c \) infinitely often even when no requests are sent, and \( \psi_3 \) is not strong enough to implement a system that in each step first produces an output and then reads the input. Assume that the system starts with output \( g = 0 \) in time step 0, then receives the input \( r = 1 \) and \( c = 0 \), now in time step 1, it can choose between (a) \( g = 1 \), or (b) \( g = 0 \). If it chooses to set grant to high by (a), then the environment can provide the same inputs once more \((r = 1 \text{ and } c = 0)\) and can set all subsequent inputs to \( r = 0 \text{ and } c = 1 \). Then the environment has satisfied \( \psi_3 \) because during the two requests in time step 0 and 1 the signal \( c \) was kept low, but the system cannot fulfill Part (i) of its specification without violating Part (ii) due to \( g = 1 \) in time step 1 and \( c = 1 \) afterwards. On the other hand, if the system decides to choose to set \( g = 0 \) by (b), then the environment can choose to set the inputs to \( r = 0 \text{ and } c = 1 \) and the system again fails to fulfill Part (i) without violating (ii). The assumption \( \psi_4 = \square(r \rightarrow \bigcirc\Diamond\neg c) \), which is a subset of \( \psi_3 \), is sufficient. However, there are infinitely many sufficient assumptions between \( \psi_3 \) and \( \psi_4 \), such as \( \psi'_3 = (\neg c \land \bigcirc \psi_3) \lor \psi_3 \). The assumption \( \psi_5 = \square(r \rightarrow \bigcirc\Diamond(\neg c \lor g)) \) is also weaker than \( \psi_3 \) and still sufficient, because the environment only needs to lower