Specifications for reactive systems often consist of environment assumptions and system guarantees. An implementation should not only be correct, but also robust in the sense that it behaves reasonably even when assumptions are (temporarily) violated. We present an extension of the requirements analysis and synthesis tool RATSY, which is able to synthesize robust systems from GR(1) specifications. A finite number of safety assumption violations is guaranteed to induce only a finite number of safety guarantee violations. We show how the specification can be turned into a two-pair Streett game, and how a winning strategy corresponding to a correct and robust implementation can be computed. Finally, we provide some experimental results.
is performed via special automata incorporating these metrics. Robustness of sequential circuits is also addressed in [7]. Inputs are divided into control and disturbance variables. A system is robust if a finite number of changes in disturbance inputs result in a bounded number of changes in the outputs. Synthesis is not addressed.

The rest of this paper is organized as follows. Section 2 presents an example to illustrate the problem. Section 3 explains our method to synthesize robust systems. Section 4 explains the computation of a winning strategy for two-pair Streett games in more detail. In section 5 our method is applied to an example. Section 6 presents experimental results and concludes.

2 Illustration of the Problem

Consider the specification of a simple arbiter for a resource shared between two clients. The input signals $r_1$ and $r_2$ are used by the clients to request access to the resource. The arbiter grants access via the output signals $g_1$ and $g_2$. The system must fulfill the following requirements. First, the system is never allowed to raise both grant signals at the same time. In LTL syntax, this can be written as $G_1 = G\neg(g_1 \land g_2)$. Second, a request has to be followed immediately by a grant, which can be formalized by the guarantees $G_2 = G(r_1 \rightarrow X g_1)$ and $G_3 = G(r_2 \rightarrow X g_2)$. Finally, it is assumed that the environment never raises both request signals at the same time: $A = G\neg(r_1 \land r_2)$. Combining the three guarantees and the assumption results in the specification $\phi = A \rightarrow G_1 \land G_2 \land G_3$. It requires the arbiter to satisfy all three guarantees, if the assumption is fulfilled.

![Synthesized Finite State Machines.](image)

One possible implementation of $\phi$ (in form of a finite state machine) is shown in Figure 1(a). If the environment assumption is violated, i.e., $r_1$ and $r_2$ are raised at the same time, the machine enters state $S_3$, and will remain there forever. Irrespective of future inputs, both grant signals stay low, therefore $G_2$ and $G_3$ will not be fulfilled anymore. This is not robust: a finite number of environment errors leads to an infinite number of system errors, i.e., the system does not recover. Our new synthesis algorithm guarantees that this cannot happen. Instead, our approach may lead to an implementation as shown in Figure 1(b), which does not exhibit the aforementioned weakness. If two requests occur simultaneously now, one will be discarded while the other one will be granted. Once the environment resumes correct behavior, the system will also fulfill all its guarantees again.
3 Robust Synthesis from GR(1) Specifications

A GR(1) specification consists of environment assumptions and system guarantees. There are two kinds of assumptions and guarantees. Safety properties encode conditions which have to hold in all time steps. They are of the form $GAp_i$ for some Boolean condition $p_i$. Fairness properties are conditions which have to hold infinitely often. They can be written as $GFp_j$. We define an implementation to be robust if a finite number of safety assumption violations induces only a finite number of safety guarantee violations.

Our robust synthesis method consists of two steps. First, the specification is transformed into a one-pair Streett game via a counting construction \cite{1}. The safety properties are encoded directly into the transition relation of the Streett game. The fairness properties are expressed via the Streett pair. For $m$ fairness assumptions $GFA_i$ (with $1 \leq i \leq m$) and $n$ fairness guarantees $GFG_j$ (with $1 \leq j \leq n$), the state-space is extended with two counters $x \in \{0, \ldots , m\}$ and $y \in \{0, \ldots , n\}$, which can be encoded with $\lceil \log_2 (m+1) \rceil + \lceil \log_2 (n+1) \rceil$ additional bits. The counter $x$ is incremented modulo $m+1$ whenever assumption $A_i$ (corresponding to the current counter value) is satisfied; similarly for $y$, $G_j$, modulo $n+1$. If a counter has the special value 0, it is always incremented. The counter value $x=0$ indicates that all $A_i$ have been satisfied in a row; $y=0$ indicates that same for all $G_j$. Hence, the condition $(GFx=0) \rightarrow (GFy=0)$, expressed by the Streett pair $\langle (x=0),(y=0) \rangle$, ensures that the liveness part of the specification is encoded properly in the game. A winning strategy for this game corresponds to a correct implementation.

In order to obtain a system which is also robust, we extend the state-space of the Streett game by two additional Boolean variables $ok_e$ and $ok_s$. Variable $ok_e$ is set to false whenever the environment violates some safety assumption, $ok_s$ is set to false iff the system violates a safety guarantee. Initially, both $ok_e$ and $ok_s$ are set to true. Our notion of robustness can now be formulated using the condition $(GF\neg ok_s) \rightarrow (GF\neg ok_e)$, which is expressed by the Streett pair $\langle (\neg ok_s), (\neg ok_e) \rangle$. An infinite number of system errors is only allowed if there is an infinite number of environment errors.

A winning strategy for the two-pair Streett game corresponds to a correct and robust implementation. We use a recursive fixpoint algorithm to compute the winning region \cite{9}. Intermediate results of this computation can be used to obtain the winning strategy.

4 Computing a Winning Strategy for Streett(2)

Figure 2 shows the algorithm to compute the winning region of a Streett game \cite{9}. The input Set is a set of Streett pairs $\langle a,b \rangle$. The function $pr(X)$ returns the set of states from which the system can force the play into $X$ in one step. $LFix$ and $GFix$ represent least and greatest fixpoint computations over sets of states. The operators $\cup$, $\cap$ and $\complement$ perform intersection, union, and complementation of sets.

The following discussion assumes $Set=\{\langle a_1,b_1 \rangle, \langle a_2,b_2 \rangle \}$. Let $Y_1$ be the fixpoint in $Y$ for the first Streett pair in the top-level call to $Str$. $Y_2$ is the result for the second pair. We denote the iterates of these fixpoint computations by $Y_{1,0} \ldots Y_{1,C_1}$ and $Y_{2,0} \ldots Y_{2,C_2}$. For both Streett pairs, the function $Str$ is called recursively. The iterates of $Y$ in the recursive call during the computation of $Y_{i,j}$ are denoted $Y_{i,j,0} \ldots Y_{i,j,C_{i,j}}$ for $i \in \{1,2\}$ and $j \in \{0, \ldots , C_i \}$.

Figure 3 illustrates the intuitive meaning of the iterates. As long as $a_1$ and $a_2$ hold, it is possible to proceed to the next lower iterate of $Y_i$. $Y_2$ is reachable from $Y_{1,1}$ and $Y_1$ is reachable from $Y_{2,1}$. The resulting cycle allows to visit $b_1$ and $b_2$ infinitely often. If $a_2$ is not satisfied, the next lower iterate of $Y_2$ may not be reachable. Not reaching $b_2$ ever again is fine if $a_2$ is also never satisfied again. However,
Figure 2: Algorithm to compute the winning strategy.

Figure 3: Illustration of the iterates of the fixpoint computation.

the other Streett pair still has to be handled. This is ensured through the iterates from the recursive step. Figure 3 shows them for $Y_{2,2}$ only. If $a_1$ holds, it is possible to proceed to the next lower iterate of $Y_{2,2}$ and from $Y_{2,2,1}$ back to $Y_{2,2}$. This cycle ensures that $b_1$ is visited infinitely often if $a_1$ holds infinitely often but $a_2$ does not. Analogously for all other iterates $Y_{i,j}$.

To define a strategy, we introduce one bit $m$ of memory. $m=0$ means $b_1$ should be fulfilled next, $m=1$ means $b_2$ should be fulfilled next. The strategy is composed of several parts, which we enumerate in the following table. They are prioritized from top to bottom. If a particular sub-strategy cannot be applied (because of violated assumptions), the next one is tried.
To demonstrate our approach, this section gives an example. Consider the specification of a full-handshake protocol with a request input signal $r$ and a grant output signal $g$. For the environment, the safety assumption $A_1 = G((r \land \neg g \rightarrow Xr) \land (\neg r \land g \rightarrow X \neg r))$ and the fairness assumption $A_2 = GF(\neg r \lor \neg g)$ are defined. The system has to satisfy the safety guarantee $G_1 = G((\neg r \land \neg g \rightarrow X \neg g) \land (r \land g \rightarrow Xg))$ and the fairness guarantee $G_2 = GF((r \land g) \lor (\neg r \land \neg g))$. Combining the assumptions and the guarantees results in the specification $\varphi = A_1 \land A_2 \rightarrow G_1 \land G_2$.

First, the specification is transformed into a one-pair Streett game. In this example there is no need for a counting construction, since there is only a single fairness assumption and guarantee. Figure 4(a) illustrates the encoding of the safety properties in the transition relation of the Streett game. The first bit of each state corresponds to the request signal $r$ and the second bit to the grant signal $g$. For example, the transitions require that, if there is a request, $r$ has to stay high until the request is granted.

Figure 4: arbiter example (a) Encoding of the safety properties in the transition relation. (b) Extension of the state space.
The following step is to **extend the state space** with the variables $ok_e$ and $ok_s$, as shown in Figure 4(b). The third bit of each state corresponds to the signal $ok_e$, which encodes an error caused by the environment. If this bit is *true*, no error occurred. Black solid lines indicate that there is no system error ($ok_s = 1$) and red dashed-lines indicate that there is one ($ok_s = 0$). Colored states represent states where an environment error has occurred. E.g., assume we start in state 101. In this state, a request occurred which has not been granted yet, and no environment error occurred. The safety assumption prohibits the environment from lowering the request. If it does anyway, depending on the choice of the system, either the state "010" or "000" is entered, which are both colored states.

Next, the **winning region and the strategy** are computed. Figure 5 illustrates the iterates of the fix-point computation. We have $a_1 = \neg (r \land g)$, $b_1 = (r \land g) \lor (\neg r \land \neg g)$, $a_2 = \neg ok_s$, $b_2 = \neg ok_e$. To illustrate strategy computation, we consider the following scenario. Assume that $m = 1$ and the arbiter is in a state out of $Y_{2,2} \backslash Y_{2,1}$. The value of $m = 1$ dictates to visit a state out of $Y_{2,1}$ next, if possible. $Y_{2,1}$ contains all states with an environment error. If we assume that the environment always behaves correctly, the set $Y_{2,1}$ becomes unreachable. In order to win the game anyway, the system is not allowed to make a mistake either, so the arbiter stays in $Y_{2,2}$. This way the second Streett pair $\langle (\neg ok_s), (\neg ok_e) \rangle$ is fulfilled, because both sets are only visited finitely often. To win the game, the first Streett pair also has to be fulfilled. Therefore the subgame is entered, trying to reach states in $b_1$ while staying in $Y_{2,2}$. Through the loop in $Y_{2,2}$, it is possible to visit these states infinitely often, fulfilling the first Streett pair as well.

Figure 5: Illustration of the iterates of the fixpoint computation.

### 6 Results and Conclusions

We tested our implementation in RATSY with an arbiter, with $N$ request and acknowledge lines (cf. Section 2). Table 6 compares the synthesis time (seconds) and the implementation size (lines of Verilog), with and without robustness. As expected, the robust approach takes more time and creates larger circuits than RATSY’s original synthesis algorithm. This is due to the higher complexity of the new method. Simulating the synthesized systems shows that the number of system errors needed to recover after one environment error is really small. In most practical cases only one or even no system errors are needed.

The original synthesis algorithm of RATSY gave no formal guarantees for robustness. The extension presented in this paper guarantees that synthesized systems are *correct-and-robust-by-construction*. This comes at the cost of larger circuits and longer synthesis times, due to the increased computational
complexity. Experimental results show that synthesized robust systems are able to recover with just very few system errors. In many practical cases, the ratio between system errors and environment errors is less than one. Since, in practice, one has to be prepared for environment errors, guaranteed robustness is an important property enhancing the quality of a system.

### References


