Designing Controllers for Reachability

Cristina Cerschi Seceleanu
Åbo Akademi University and Turku Centre for Computer Science (TUCS),
Lemminkäisenkatu 14, FIN-20520 Turku, Finland
ccerschi@abo.fi

Abstract

We propose a deductive method for constructing reliable reachability controllers, with application to fault-tolerant discrete systems. Designing the controller reduces to finding a strategy to win specific games defined by sequential angelic and demonic nondeterministic statements. During the game, the plant (the demon) tries to prevent the controller (the angel) from achieving its respective goal, modeled by a special kind of liveness property. We show that the angel has a way to enforce the required property, provided that adequate invariance and termination properties hold. The control strategy is obtained by propagating certain assertions into the angelic statement. We illustrate our method on a data-processing application.

1. Introduction

Many reactive systems are actually control systems, thus it is essential to find a rigorous way of constructing their controllers. The interaction between the controller and its environment can be modeled as a two-player game on a possibly infinite state space.

In this paper, we propose such a game-based method for solving the problem of reachability control, non-algorithmically. Our work uses refinement calculus [4, 11] as the reasoning framework, and action systems [2] as the modeling language. We describe the discrete reactive system as a game between two rival agents, the angel and the demon. Not surprisingly, the angel models the controller, and the demon represents the plant, or the disturbance. The system model is in fact an action system with two kinds of nondeterministic statements known as angelic updates and demonic updates.

Our method starts by checking whether the angel has a way to win the game with respect to the specified requirement, which is modeled by a newly defined temporal property. The main temporal properties have been formalized within the extended weakest precondition framework, by Back and von Wright [6, 7].

Since we address synthesis of reliable controllers for reachability, the requirement is a variant of the eventually property, which we call weak response. The angelic controller has to enforce the weak response property, if it can, regardless of the actions of the demonic plant. To accomplish this, the angel has to find a way to guide the system into a target predicate, in finite time, provided that some associated condition holds. After that, the execution terminates. Alternatively, in order to win, the angel can keep the system outside the associated condition, forever.

We show that enforcement of weak response properties is reduced to traditional correctness properties of a special fixed point statement. Hence, we propose an inference rule that can be used for checking the existence of an angelic winning strategy, for weak response games. If such a winning strategy does exist, we extract it, next, by rewriting the respective angelic statement, in a certain context resulted from the correctness proofs carried out in the first step. This transformation restricts the choices of the angel, such that all of those that, if taken, would violate the requirement, are eliminated from the angelic statement. The result is a correct lower-level model, which is guaranteed to preserve the required temporal property. We apply our method to construct the controller of a fault-tolerant reactive system.

Related Work. Ramadge and Wonham [13], and Pnueli and Rosner [12] developed synthesis algorithms for finite-state reactive games, and showed that finding a winning strategy for the game was equivalent to synthesizing a controller that satisfied the requirements.

Tripakis and Altisen have also proposed algorithms to solve the issue of controller synthesis for finite-state discrete and dense-time systems [15]. Their algorithms are fully on-the-fly, that is, a strategy is returned as soon as it is found, thus the state space does not necessarily have to be entirely generated.

Asarin et al. apply game techniques to construct discrete controllers, by means of fixed-point algorithms [1]. Being deductive, our approach can also handle infinite-state systems as such, without requiring their finite abstractions.
2. Background

In this section, we give an overview of statements and introduce action systems. The notation and the main concepts are taken from previous work of Back and von Wright [4, 5, 6, 7].

Our reasoning framework, refinement calculus, uses higher-order logic as the underlying logic. A statement \( S \) is built according to the syntax below:

\[
\langle f \rangle \; \{ p \} \; | \; \{ p \} \; | \; S_1 ; S_2 \; | \; S_1 \sqcap S_2 \; | \; \{ x := x' \; | \; b \} \; | \; \{ x := x' \; | \; b \}
\]

Here, \( p \) ranges over state predicates (\( \Sigma \to \text{ Bool} \)), \( f \) over state transformers (\( \Sigma \to \Sigma \)), and \( x := x' \; | \; b \) is a state relation (\( \Sigma \to \Sigma \to \text{ Bool} \)), where \( \Sigma \) is the polymorphic type of the program state. We write \( f.x \) for function \( f \) applied to \( x \).

The functional update, \( \langle f \rangle \) changes the state according to the state transformer \( f \) (for example, \( \{ x := e \} \) is a special kind of update where the state transformer is expressed as an assignment). We use the name \texttt{skip} for the identity update. The assertion \( \{ p \} \) leaves the state unchanged if \( p \) holds and aborts otherwise, whereas the \texttt{assumption} \( [p] \) also leaves the state unchanged if \( p \) holds, but terminates miraculously otherwise (i.e., it establishes any postcondition, even false).

The angelic nondeterministic assignment (or angelic update), \( \{ x := x' \; | \; b \} \), lets the controllable angel to choose the final state, among those that satisfy the boolean condition \( b \), whereas in the demonic nondeterministic assignment (or demonic update), \( x := x' \; | \; b \), the choice is demonic ( uncontrollable). If no such state exists, then the angelic update is aborting (i.e., it establishes no postcondition, not even true), while the demonic update is miraculous. A sequence of an angelic and a demonic update is interpreted as a game with the angel and the demon as players: \( \{ x := x' \; | \; b \} ; \{ x := x' \; | \; b \} \).

The demonic choice, \( S_1 \sqcap S_2 \), is treated similarly to the demonic update, that is, the demon chooses, nondeterministically, to carry out \( S_1 \) or \( S_2 \).

A predicate transformer is a function that maps predicates to predicates. We want the predicate transformer \( S \) to map postcondition \( q \) to the set of all initial states \( \sigma \) from which \( S \) is guaranteed to end in a state of \( q \). Thus, \( S.q \) is the weakest precondition (w.p.) of \( S \) to establish postcondition \( q \). A predicate transformer \( S \) is monotonic if: \( p \subseteq q \Rightarrow S.p \subseteq S.q \). The intuitive description of statements can be used to justify the following definition of the weakest precondition semantics:

\[
\begin{align*}
(S_1 ; S_2).q &= S_1.(S_2.q) \quad (1) \\
\{ x := x' \; | \; b \}.q &= ((\exists x' \cdot b \wedge q[x := x']) \quad (2) \\
[x := x' \; | \; b].q &= (\forall x' \cdot b \Rightarrow q[x := x']) \quad (3)
\end{align*}
\]

These definitions are consistent with Dijkstra’s original semantics for the language of guarded commands [8], and with later extensions to it. We say that the angel has a strategy to win an angel-demon game, if and only if the angel has a way of making its choices inside \( S \) such that the predicate \( q \) holds in the final state, regardless of how the demon makes its choices.

Our language also permits recursive statements, in form of \( (\mu X \cdot S) \) or \( (\nu X \cdot S) \), depending on whether statement \( X \) can be invoked by the angel a finite number of times, or infinitely, respectively. An important particular case of recursion is the \texttt{do – od} loop, which is defined in the usual way: \( \texttt{do } g \to S_1 \texttt{ od } \triangleq (\mu X \cdot \texttt{if } g \texttt{ then } S_1 \texttt{ else skip} ) \).

In this paper, we consider a special form of an action system, as follows:

\[
\text{Sys}(y) \triangleq \text{begin var } x \cdot S_0 ; \texttt{do } g \to A ; D \texttt{ od end} \quad (4)
\]

Here, \( \text{Sys} \) contains an initialisation statement \( S_0 \) and the action statement \( S \), which has a guarded form, \( S = g \to A ; D \). The boolean condition \( g \) is the action guard and \( A ; D \) is the action body (\( A \) is an angelic statement and \( D \) a demonic one). The initialization statement typically introduces some local variables, \( x \), for the action system, and initialises these. Variables \( y \) are global to the action system, and they are also assigned initial values by \( S_0 \). The action \( S \) is enabled, thus its body \( A ; D \) is executed, when the guard \( g \) holds. Termination is normal if the exit condition \( \neg g \) holds.

The predicate transformer semantics is based on total correctness. Here, \( p \; \langle \{ S \} \rangle q \equiv p \subseteq S.q \) denotes the total correctness of statement \( S \) with respect to precondition \( p \) and postcondition \( q \).

We say that \( S \) is refined by \( S' \), written \( S \subseteq S' \), if and only if \( (\forall q \cdot S.q \subseteq S'.q) \).

A refinement rule is an inference rule that allows us to deduce that a certain refinement \( S \subseteq S' \) is valid. Adding choices to an angelic update and removing choices from a demonic update are both valid refinements. Equality “=” of statements is a stronger form of refinement.

We will use the rule of \texttt{backward propagation} of assertions into angelic nondeterministic assignments:

\[
\{ x := x' \; | \; b \} ; \{ q \} = \{ x := x' \; | \; b \wedge q[x := x'] \} \quad (5)
\]

3. Synthesis of Controllers for Reachability

Reachability controllers have to guide the system into a specified set of states, from any initial state, or provided that some nontrivial condition is fulfilled. One possible occurrence of the latter is in fault-tolerant systems.

Even if we agree with the point of view that one of the best ways to deal with faults is not to have them, it is not possible to make any system 100% fault-free. Failures always occur, resulting from many causes, such as degradation, overloading, design errors etc. Besides minimizing the chances of a failure, one should also strive to reduce its effect. This means that a system has to be designed both as a fault-tolerant system and also as a fault-preventive one.
For example, one may wish to encode the requirement that a system failure is eventually followed by a return to the normal mode of operation (fault - tolerance), but it is not followed by other failures (fault - prevention).

3.1 Characterizing Enforcement of Response Properties in Action Systems

Formal Definition of Weak Response. Let us consider a reactive system described by an action system as in (4). In principle, for the design of reachability controllers, the angel has to guarantee liveness properties, modeled as “eventually” ($\Diamond$) properties. Here, we focus on an eventuality property that we denote by the temporal operator $\Diamond_w(p,q)$, $p, q$ predicates. We call this property weak response.

The weak response property holds if, for any state in the set of reachable states $p$, the angel has a way to lead the system into a state of $q$, in finite time, after which the execution terminates, or if the angel is at least able to keep the system into a state of $\neg p$, forever.

As proved by Back and von Wright [6], for “always” and “until” properties, enforcement of temporal properties is reduced to traditional correctness properties of special fixed-point statements. To be able to further characterize $\Diamond_w(p, q)$, we define the following recursive statement, which subsumes the existence of two loops:

$$\text{WRes.p.q} \equiv (\nu X \cdot [\neg p] ; [g] ; A ; D ; X \cap [p] ; (\mu Y \cdot [\neg q] ; \{g\} ; A ; D ; Y))$$

(6)

In fact, WRes.p.q is a weak iteration that is necessarily terminating, if the particular condition $p$ holds. Observe that the exit alternative of the recursion is not skip, like traditionally, therefore our definition (6) resembles tail recursion, as defined by Back and von Wright [4].

We can say that WRes.p.q is an interpreter for $\Diamond_w(p,q)$, which executes the constituent statements for determining whether the weak response property is valid. The behavior of WRes.p.q is shown in Figure 1 ($\top$ symbolizes a win).

For example, $p$ may model a (permanent) fault in the functioning of the system. Thus, we would like the system to behave correctly and reliably, even after that fault had occurred. However, the performance of the system is maintained for a limited time.

As $p$ abstracts a fault or a fixed number of faults ($n$), termination is compulsory after $n$ has been encountered, since the system can not tolerate more than an $n$-error scenario. Hence, in this case, $q$ needs to be established prior to, or at most at the same time with termination.

We translate these informal descriptions into a formal lemma, which characterizes the weakest precondition for a statement to establish the weak-response property.

Lemma 1 Assume that predicates $p, q$, and the monotonic predicate transformer $S$ are given. Then

$$S.\Diamond_w(p,q) = (\nu X \cdot (p \cup S.x) \cap (\neg p \cup (\mu Y \cdot q \cup S.y)))$$

The characterization in Lemma 1 shows how we can compute the weakest precondition of a predicate transformer, to establish weak response, by fixed point computation of predicates.

Correctness Lemma for Weak Response. Here, we formulate enforcement of weak response, as a correctness property.

Since

$$\forall \sigma \cdot \sigma \{ \{\text{do } g \rightarrow A ; D \text{ od } \} \Diamond_w(p,q) \equiv (\text{WRes.p.q}).\text{false. } \sigma,$$

we can further claim the following result.

Lemma 2 Let statements $A, D$, and predicates $p, q$ be the same as above. Then

$$p_0 \{ \{\text{do } g \rightarrow A ; D \text{ od } \} \Diamond_w(p,q) \equiv p_0 \subseteq (\nu X \cdot [\neg p] ; [g] ; A ; D ; X \cap [p] ; (\mu Y \cdot [\neg q] ; \{g\} ; A ; D ; Y)).\text{false. }$$

Lemma 2 proves that we can reduce the question of whether weak response can be enforced for an action system, to the question of whether a certain goal can be achieved. In this case, the goal false cannot really be established, so success can only be achieved by miraculous termination, or by non-termination caused by the demon.

3.2 Proving Enforcement of Weak Response

Invariant-based Proof Rule. Below, we propose an inference rule for proving enforcement of weak-response properties.
Lemma 3 Assume the following action system:

$$\text{Sys}(y : T_y) = \begin{align*}
\text{begin} & \quad \text{var } x : T_x \cdot S_0; \\
& \quad \text{do } g \rightarrow A ; D \text{ od end}
\end{align*}$$

Then, weak-response properties can be proved using invariants, and termination arguments, as follows:

$$
\begin{array}{c}
p_0 \subseteq g \cap I \\
p \cap I \subseteq q \cup g \\
\neg q \cap g \cap (t = w) \subseteq ([A ; D] \cap (q \cup g) \cap (t < w))
\end{array}
$$

Here, $p$, $q$ are predicates, and the state function $t$ ranges over some well-founded set.

The rule of Lemma 3 shows what proof obligations we have when carrying out controller construction for weak response. The predicate $I$ might include states of $q$, or states of $\neg p$. Therefore, the angel is considered to have a winning strategy if it can find a way to keep the system in $I$, and lead the system into a state of $q \cup g$, trying to decrease $t$, provided that it has started from a state of $\neg q \cap g$. This means that the controller is forced to make appropriate moves, such that at the end of the game, $q \cup g$ gets established, after a finite number of iterations ($t < w$).

3.3 Extracting the Control Strategy

After having established that the angel can enforce the required behavior, the next step is to extract its respective winning strategy.

In the following, we show how to reduce the angelic nondeterminism, with respect to the enforced property. This is achieved by finding a statement $A'$ that contains fewer angelic choices than $A$.

Given the fact that $I$ is an invariant of the action system $\text{Sys}$, defined in Lemma 3, we know, due to the weakest precondition rule for sequential composition (1), that statement $S = \{I : A ; D.I \} ; D ; \{I\}$ can replace the sequence $A ; D$ of the $\text{do } - \text{od}$ loop, since $S$ preserves the invariant, trivially. In consequence, we can rewrite $A$ by using the information supplied by $\{D.I\}$, such that the angel is forced to restrict its choices to the ones that establish $I$.

Let us consider $A = \{x := x' \mid b\}$. Then, $A' = \{x := x' \mid b \cup D.I\}$. Next, we use the assertion $\{D.I\}$ to rewrite $A'$, hence we propagate it backwards. In this way, we strengthen the boolean condition of the angelic nondeterministic assignment. As a result, the angelic choices are reduced, according to the propagated information. We apply the refinement rule (5), as follows:

$$\{x := x' \mid b\} ; \{D.I\} = \{x := x' \mid b \land D.I[x := x']\} \quad (7)$$

In principle, this is a transformation that does not favor the angel, it rather makes the demon happy, since it decreases the set of final states that the angel can choose from. However, rewriting in the specified context makes the behavior of the angel amenable to automation.

The proof of $\neg q \cap g \cap (t = w) \subseteq A.(D.((q \cup g) \cap (t < w)))$ shows only termination and establishment of $q$ in finite time. The information given by $\{D.I\}$ is sufficient for extracting the angelic winning strategy.

4. Example: A Data Processing System

In this section, we analyze the operation of an abstract, distributed data processing system. The input data is produced by one unit (PU), and transferred, via a limited capacity channel (buffer), to a collection of collector devices (CD) that process it further. The channel is similar to a buffer that contains $\text{cap}$ locations. A graphical representation of the system is given in Figure 2.

![Figure 2. The data processing system](image-url)
are helped in this quest by the fact that, built with its own safety considerations, the controller receiving the target address from the PU, may reconstruct it, once, during the life time of the system, even though the necessary data is missing, or corrupted.

Based on the just mentioned fault-tolerant feature, one may think of an optimized system that works “normally”, until it falls in the undesired erroneous state, from where the address of the next processing CD may not be extracted by the monitor. If this is the first time when the event occurs, the error is detected and repaired by the monitoring procedures. Following this, the system must be protected from reaching again the same situation, during its life time. The resulting system is illustrated in Figure 3.

![Figure 3. The data processing system, with optimized dimensions](image)

The behavior of the optimized system can be split into two operation modes. Firstly, if there is no conflict regarding the content of the last buffer element, the safety requirement of keeping the content of the buffer at most \( cap-1 \) ensures the correct functionality of the system. Additional measures must be imposed if a conflict appears; the first time must also be the last time when such a situation arises, during the system’s life time. In short, if for the optimized system one can find a strategy that not only ensures that there is no overflow of the buffer, but also accommodates the new requirements (the target address saved in the buffer, too), the system will evolve at the parameters presented previously. However, if such a strategy does not exist, the designer should allow one error of the conflicting kind, but, after this error has been accounted for, a second similar event must be avoided.

**System Modeling.** The set-up of our example takes into account a system composed of one PU module (the controller) and a collective CD module (the plant). The PU is the angelic subsystem, while the CD constitutes the demonic part of the system. Hence, the incoming signals are modeled as angelic updates, whereas the removal of the buffer content is viewed as a disturbance. The capacity of the buffer between the two components is represented by the variable \( cap \) (\( cap \geq 4 \)). The execution of the composed system develops in rounds, where every update performed by the angel is followed by one carried out by the demon. The end of an execution round is marked by incrementing a variable \( (life : \text{Nat}) \) that records the functioning time of the system.

The angel, PU, adds one or two data elements to the buffer (apart from the newly required target address). The number of packages in the buffer is modeled by the natural variable \( C \). Whenever its value goes over the value \( cap-1 \), the system is able to maintain the data content of the buffer within the limits \( 0 \) and \( cap-1 \). Thus, the system avoids the appearance of a second, similar error, and \( err = 1 \land life = \text{LimFunc} \) is enforced.

The behavior of the demon follows a statistically determined, partly nondeterministic pattern. Firstly, the CD may remove one or two packages from the buffer, or leave it untouched. Secondly, the removal pattern is described by the following specification: if in the previous round no package has been removed, then, in the present one, CD removes two packages; if, in the previous round, one package has been removed, in the present round, the CD may leave the buffer unchanged, or remove one or two packages; the CD module will remove a single data package if in the previous round it has processed two packages from the buffer.

The angel wins in two situations:

1. if the system reaches the end of its life time (expressed in the model by the constant \( \text{LimFunc} \)) and there has not been any conflict between placing the data packages and the target address, that is, \( err = 0 \land life = \text{LimFunc} \) holds.

2. if, after one error has been signaled, the system is able to maintain the data content of the buffer within the limits \( 0 \) and \( cap-1 \). Thus, the system avoids the appearance of a second, similar error, and \( err = 1 \land life = \text{LimFunc} \) is enforced.

The angel loses in the following situation:

1. if, after one error has been signaled, the system is not able to maintain the data content of the buffer within the limits \( 0 \) and \( cap-1 \). This means that a second conflict target address - data input can not be avoided.

Also, at any time during valid operation (\( life < \text{LimFunc} \)), we would like the angel to avoid leaving the demon in the impossibility to take any package from the buffer. In other words, if the angelic update establishes \( C = 0 \) and the demon has the possibility, or the duty to
remove at least one element from the buffer, we run into miraculous behavior, which we forbid.

The behavior of the angel is represented by the statement PU:

$$\text{PU} \triangleq \{ C := C' | C < C' \leq C + 2 \}$$

(8)

The demonic behavior is captured as follows:

$$\text{CD} \triangleq (C \geq \text{cap} \land \text{err} = 0 \rightarrow \text{err} := \text{err} + 1; \ C := C - k)$$
$$\quad \land C \geq \text{cap} \land \text{err} > 0 \rightarrow \text{err} := \text{err} + 1$$
$$\quad \land C < \text{cap} \rightarrow \text{skip};$$
$$\quad [ r := r' | (r = 0 \Rightarrow r' = 2)$$
$$\quad \land (r = 1 \Rightarrow r' \in \{0, 1, 2\})$$
$$\quad \land (r = 2 \Rightarrow r' = 1)];$$
$$\quad C := C - r;$$
$$\quad \text{life} := \text{life} + 1$$

The additional information that we need is given as:

$$q \triangleq (\text{err} = 1) \land (\text{life} = \text{LimFunc})$$
$$g \triangleq \text{life} \leq \text{LimFunc} - 1$$
$$p \triangleq (\text{err} > 0) \land (\text{life} < \text{LimFunc})$$

(9)

The invariant $I$ ensures that, in case there exists a winning strategy, we obtain a system where the capacity of the buffer is not exceeded, which is in turn used in the "thinking" angelic process intended to rule out possible erroneous moves. The current invariant also takes into account the presence of the error messages.

The overall PU-CD system is then described as the following action system game:

$$\text{DPS} \triangleq \begin{align*}
\text{begin} & \text{ var } r \in \{0, 1, 2\}, \\
& \quad C, \text{life, err} : \text{Nat} * \\
& \quad r, C, \text{life, err} := 1, 4, 0, 0; \text{cap} := 8; \\
& \quad \text{do} \quad (10) \\
& \quad \text{life} \leq \text{LimFunc} - 1 \rightarrow \text{PU} ; \text{ CD} \\
& \quad \text{od} \\
& \text{end}
\end{align*}$$

Applying the Synthesis Method for Weak Response. In order to find a winning strategy for the angel (if any), we have to first check the validity of the following relations.

1. $p_0 \subseteq I$. Immediate proof, after replacing $I$ with its definition (9) and also considering $p_0 = (\text{err} = 0 \land r = 1 \land C = 4 \land \text{life} = 0 \land \text{cap} = 8)$.

2. $p \cap I \subseteq q \cup g$. From definitions (9) we have, at first, that

$$q \cup g = (\text{err} = 1) \land (\text{life} = \text{LimFunc})$$
$$\quad \land (\text{life} \leq \text{LimFunc} - 1)$$

The proof of the required relation is immediate.

3. $g \cap I \subseteq \text{PU.(CD.I)}$. We start by denoting:

$$\text{CD}1 = C \geq \text{cap} \land \text{err} = 0 \rightarrow \text{err} := \text{err} + 1; \ C := C - k$$
$$\quad \land C \geq \text{cap} \land \text{err} > 0 \rightarrow \text{err} := \text{err} + 1$$
$$\quad \land C < \text{cap} \rightarrow \text{skip};$$
$$\text{CD}2 = [ r := r' | (r = 0 \Rightarrow r' = 2)$$
$$\quad \land (r = 1 \Rightarrow r' \in \{0, 1, 2\})$$
$$\quad \land (r = 2 \Rightarrow r' = 1)];$$
$$\quad C := C - r;$$
$$\quad \text{life} := \text{life} + 1$$

We compute:

$$\text{CD1} = \{\text{rule (1)}\}$$
$$\text{CD1.(CD2.I)} = \{\text{rules (1), (3), logic}\}$$
$$\quad \text{CD1.}((\text{r} = 0 \Rightarrow \text{err} = 0 \land (2 \leq C \leq \text{cap}))$$
$$\quad \land (r = 1 \Rightarrow \text{err} = 0 \land (2 \leq C \leq \text{cap} - 1))$$
$$\quad \land (r = 2 \Rightarrow \text{err} = 0 \land (1 \leq C \leq \text{cap} - 1))$$
$$\quad \land (r = 0 \Rightarrow \text{err} = 0 \land (2 \leq C \leq \text{cap} - 2))$$
$$\quad \land (r = 2 \Rightarrow \text{err} = 0 \land (1 \leq C \leq \text{cap} - 2)))$$
$$\quad = \{\text{logic}\}$$
$$\text{CD1.}((2 \leq C \leq \text{cap} - 2)$$
$$\land (\text{err} = 0) \land (C = \text{cap} - 1)$$
$$\land (\text{err} = 0) \land (r = 0) \land (C = \text{cap})$$
$$\land (r = 0) \land (C = \text{cap} - 1)$$
$$\land (r = 2) \land (C = 1))$$
$$\quad = \{\text{logic}\}$$
$$\text{CD1.}((2 \leq C \leq \text{cap} - 2)$$
$$\land (\text{err} = 0) \land (C = \text{cap} - 1)$$
$$\land (\text{err} = 0) \land (r = 0) \land (C = \text{cap})$$
$$\land (r = 0) \land (C = \text{cap} - 1)$$
$$\land (r = 2) \land (C = 1))$$
$$\quad = \{\text{notation}\}$$
$$\text{CD1.I_2} = \{\text{wp rules}} \{4\}$$
$$\land ((C \geq \text{cap}) \land (\text{err} = 0) \Rightarrow$$
$$\quad I_2[\text{err} := \text{err} + 1, C := C - k])$$
$$\land ((C \geq \text{cap}) \land (\text{err} > 0) \Rightarrow$$
$$\quad I_2[\text{err} := \text{err} + 1])$$
$$\land ((C < \text{cap}) \Rightarrow I_2$$
$$\quad = \{\text{logic}\}}
We can write further that:

\[
\begin{align*}
&\text{{PU.}}(\text{CD}.1.I_2) \\
&\quad = \{\text{definition, rule (2)}\} \\
&\quad \exists C' \cdot (C < C' \leq C + 2) \land \text{CD}.1.I_2[C := C'] \\
&\quad \supseteq \{\text{case split}\} \\
&\quad \bullet \{\text{witness } C' = C + 1\} \\
&\quad (C' = C + 1) \land (2 \leq C' \leq cap - 2) \\
&\quad \forall (r = 0) \land (C' = cap - 1) \\
&\quad \forall (r = 2) \land (C' = 1) \\
&\quad \forall (err = 0) \land (C' = cap - 1) \\
&\quad \forall (err = 0) \land (C' = cap) \\
&\quad \land (k + 2 \leq C' \leq cap + k - 2) \\
&\quad \supseteq \{2 \leq k \leq cap - 2\} \\
&\quad I \cap (C \geq 2) \\
&\quad \bullet \{\text{witness } C' = C + 2\} \\
&\quad (C' = C + 2) \land (2 \leq C' \leq cap - 2) \\
&\quad \forall (r = 0) \land (C' = cap - 1) \\
&\quad \forall (r = 2) \land (C' = 1) \\
&\quad \forall (err = 0) \land (C' = cap - 1) \\
&\quad \forall (err = 0) \land (C' = cap) \\
&\quad \land (k + 2 \leq C' \leq cap + k - 2) \\
&\quad \supseteq \{\text{cap} \geq 4\} \\
&\quad I \cap (0 \leq C \leq 1) \\
&\quad = \{\text{logic}\} \\
&\quad I \cap (0 \leq C \leq 1) \cup I \cap (C \geq 2) \\
&\quad = \{\text{logic}\} \\
&\quad I \\
&\quad \supseteq \{\text{logic}\} \\
&\quad g \cap I
\end{align*}
\]

Notice the extraction of the parameter \(k\): its range of values is limited to the interval \([2, cap - 2]\). Concluding, the above derivation shows that \(I\) is, indeed, an invariant of the system, that is, \(g \cap I \subseteq \text{PU.}(\text{CD}.I)\).

4. \(\neg q \land g \land (t = w) \subseteq \text{PU.}(\text{CD}.((q \lor g) \land (t < w))).\)

We consider that \(t \triangleq \text{LimFunc} - \text{life}\). We denote:

\[
\begin{align*}
&\quad (q \lor g) \land (t < w) \\
&\quad = \{\text{definitions (9)}\} \\
&\quad (((err = 1) \land (\text{life} = \text{LimFunc}) \\
&\quad \lor (\text{life} \leq \text{LimFunc} - 1)) \\
&\quad \land (t < w) \}
&= \{\text{definition of } t\} \\
&\quad (((err = 1) \land (\text{life} = \text{LimFunc}) \\
&\quad \lor (\text{life} \leq \text{LimFunc} - 1)) \\
&\quad \land (\text{life} \geq \text{LimFunc} - w + 1) \\
&\quad \land (2 \leq C \leq cap - 2) \\
&\quad \lor (\neg q \land g \land (t = w)) \\
&\quad \land (C = cap - 1) \\
&\quad \lor (\neg q \land g \land (t = w)) \\
&\quad \land (C = 1) \\
&\quad \lor (\neg q \land g \land (t = w)) \\
&\quad \land (\text{life} \leq \text{LimFunc} - 1) \\
&\quad \lor (\neg q \land g \land (t = w)) \\
&\quad \land (\text{life} \geq \text{LimFunc} - w + 1)
\end{align*}
\]

We then have that:

\[
\begin{align*}
&\quad \text{PU.}(\text{CD}.((q \lor g) \land (t < w))) \\
&\quad = \{\text{definitions (9), notation (11)}\} \\
&\quad \text{PU.}(\text{CD}.X) \\
&\quad = \{\text{rule (1)}\} \\
&\quad \text{ PU.}(\text{CD}.1.(\text{CD}2.X)) \\
&\quad = \{\text{wp rules}\} \\
&\quad \text{PU.}(\text{CD}.1.(((\text{err} = 1) \\
&\quad \lor (\text{life} \leq \text{LimFunc} - 2)) \\
&\quad \land (\text{life} \geq \text{LimFunc} - w) \\
&\quad \lor (\text{life} \leq \text{LimFunc} - 2) \\
&\quad \land (\text{life} \geq \text{LimFunc} - w)) \\
&\quad = \{\text{rule (2)}\} \\
&\quad \exists C', (C < C' \leq C + 2) \\
&\quad \land ((\text{err} = 0) \land (\text{life} = \text{LimFunc} - 1) \\
&\quad \land (\text{life} \geq \text{LimFunc} - w) \\
&\quad \lor (\text{life} \leq \text{LimFunc} - 2) \\
&\quad \land (\text{life} \geq \text{LimFunc} - w)) \\
&\quad \supseteq \{\text{logic}\} \\
&\quad (\text{err} = 0) \land (\text{life} = \text{LimFunc} - 1) \\
&\quad \land (\text{life} \geq \text{LimFunc} - w) \\
&\quad \lor (\text{life} \leq \text{LimFunc} - 2) \\
&\quad \land (\text{life} \geq \text{LimFunc} - w) \\
&\quad \supseteq \{\text{logic, w} \geq 2\} \\
&\quad \neg((\text{err} = 1) \land (\text{life} = \text{LimFunc})) \\
&\quad \land (\text{life} \leq \text{LimFunc} - 1) \\
&\quad \land (\text{life} = \text{LimFunc} - w)
\end{align*}
\]

**Extracting the Control Strategy.** We have proved that \(I\) is an invariant of the action system \(\mathcal{DP}_8\), defined by (10). In consequence, the statement \(\{I\}; \text{PU}; \{\text{CD}.I\}; \text{CD}.(I)\) can substitute \(\text{PU}; \text{CD}\) and become the new body of the loop of \(\mathcal{DP}_8\). In consequence, we can rewrite \(\text{PU}\), by pulling the assertion \(\{\text{CD}.I\}\) through statement \(\text{PU}\). If we assume \(k = 2\), we get the winning strategy of the angel, with all the unsafe moves eliminated:

\[
\begin{align*}
&\quad \text{PU}' = \{C := C' | (C < C' \leq C + 2) \\
&\quad \land (2 \leq C' \leq cap - 2) \\
&\quad \lor (\neg q \land g \land (t = w)) \\
&\quad \lor (r = 0 \land C' = cap - 1) \\
&\quad \lor (r = 2 \land C' = 1) \\
&\quad \lor (err = 0 \land cap - 1 \leq C' \leq cap))
\end{align*}
\]

This strategy does not require any angelic intelligence in deciding the “good” moves, yet it does not rule out the non-harmful nondeterminism. Therefore, whatever choice we
select for implementation, such that the boolean condition of the assignment is satisfied, the correctness of the controller is guaranteed.

5. Conclusions

We have tackled the problem of reachability controller construction, by modeling the discrete system as an action system, and the synthesis process as a zero-sum two-player game. The players are the controller, called the angel, and the plant, called the demon, which make moves sequentially, according to some nondeterministic statement, respectively. The goal of the angel (the requirement specification) is a newly defined eventuality temporal property that we have named weak response.

To express weak response in our framework, we have introduced the temporal operator ♦w(p, q), and have characterized it formally. Proving enforcement of ♦w(p, q) reduces to the correctness proofs given in Lemma 3. We have applied the theoretical results to design a reliable reachability controller for a data processing system. The respective system is fault-tolerant, covering one-fault scenarios.

Our work relies on the angel-demon game formalization in the weakest precondition framework, introduced by Back and von Wright [3], and on its later extensions [5, 6, 7].

The synthesis subsumes two main steps. Firstly, one needs to check whether the angel can enforce the required behavior. To accomplish this, one has to discharge the proof obligations of Lemma 3. If this first step is passed, extracting the safe set of strategies, or the specific control strategy follows. This is done by backward propagation of assertions, which is a correctness-preserving transformation.

Perhaps the closest work to ours is proposed by Slanina [14], who develops proof rules for safety and response linear temporal logic properties of reactive system games. However, the equivalent of our second synthesis step, that is, extracting the angelic winning strategy, is not apparent. We are just promised that, if the control conditions can be proved valid, by invoking constructive theorem proving methods, the extracted program can be used further to synthesize a control winning strategy.

Distinctly from the fixed-point symbolic synthesis algorithms [1, 9, 10], which are expensive in memory resources, our game-based method is fit for interactive theorem proving (PVS, HOL etc.).

References


