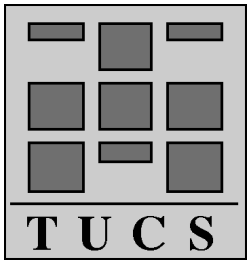
Games-based Controller Synthesis for Discrete Systems

Ralph-Johan Back
Cristina Cerschi Seceleanu



Turku Centre for Computer Science

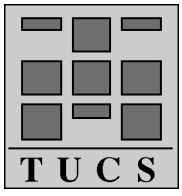
TUCS Technical Reports

No 594, October 2004

Games-based Controller Synthesis for Discrete Systems

Ralph-Johan Back
Cristina Cerschi Seceleanu

e-mail: {backrj, ccerschi}@abo.fi



Turku Centre for Computer Science
TUCS Technical Report No 594
October 2004
ISBN 952-12-1313-2
ISSN 1239-1891

Abstract

This study proposes a method for constructing reliable controllers for arbitrarily large discrete systems. The controller is synthesized by finding a winning strategy for specific games defined by contracts. The discrete system model is an action system, and the requirement is a temporal property. We use the extended action system notation that allows both angelic and demonic nondeterminism, such that the game reduces to a competition between the angel, that is, the controller, and the demon, that is, the plant, which try to prevent each other from achieving their respective goals. If the synthesis is possible, i.e., if the angel has a way to enforce the required property, the process ends with the extraction of the angelic winning strategy, by propagating certain assertions into the contract that models the controllable actions. The technique leads to a correct-by-construction program. We illustrate our method on a producer-consumer application.

TUCS Laboratory
Software Construction Laboratory

1 Introduction

Controller synthesis amounts to developing a framework for designing controllers that meet the requirements. Although there are several solutions to solve this task for discrete systems, most of them employ algorithmic techniques [1, 10]. Model-checking, as an exhaustive verification method, has proved efficient for small systems, but rather difficult to apply for large systems, as it is expensive in machine resources.

In general, *verification* requires a complete system model, often deterministic, which is verified toward satisfying a set of properties. In contrast, *synthesis* starts with an open model of the system, possibly nondeterministic. This model acts as the high-level system description, useful when the designer deals with complex requirements.

Our contribution stays mainly in constructing a method for designing reliable discrete control software for arbitrary systems, starting from a nondeterministic model.

We address the synthesis of the controller by representing the system as a game between two players, the controller, called *the angel*, and the plant, called *the demon*. Each of the players tries to achieve some specific goal at the end of the game, and at the same time tries to prevent the other from establishing its respective goal. Hence, controller synthesis reduces to finding a strategy for the angel to carry out control events, such that its goal is guaranteed in spite of the nondeterministic demonic moves. The high-level model is an *action system* [2] that allows each player to take turns and sequentially make choices that determine the next state of the system. The choices are regulated by a *contract* [3]. Back and von Wright defined temporal properties in the extended predicate transformer framework [6]. We adapt their result and capture the requirement, or the goal of the angel, as a safety property, modeled by an "*always*" ($\Box$) temporal property. We show that the property holds, by proving an invariant. In principle, the safety property does not hold for every possible execution of the system, but it should be enforceable by the angel. This is the first step of the synthesis, that is, checking whether the angel is able to play such that it enforces the required behavior. In fact, this leads to the synthesis of controllers for invariance (that have to keep the system inside a safe set of states). Controllers for reachability (that have to lead the system to a set of desired states) can also be synthesized within our framework. We introduce a proof rule that supports this claim.

If we pass the first step described above, we aim further to extract and implement the respective solution. We show that moving toward an implementation of the angel, with respect to the enforced property, reduces to decreasing, or sometimes even eliminating its nondeterminism, with regard to that property. Pursuing our goal, we rewrite the angelic contract by propagating backwards, the computed

weakest precondition of the demon, with respect to the proved invariant. In this way, we remove the angelic choices that, if taken, would violate the invariant. Thus, we force the angel to choose only from the possibilities given by the propagated information. Hence, we derive a control strategy, or, in some cases, a wrapper of all possible control strategies, which guarantees a win for the angel with regard to the property that we have considered, no matter what moves it makes during the game. All this time, the behavior of the demon remains unchanged. We work in the framework of the *refinement calculus* [4, 12], hence, on the way, we apply specific rules that guarantee the correctness of the extracted strategy.

Viewing a reactive system as a two-player game is not a new idea, it can be traced back to Ramadge and Wonham [14], and Pnueli and Rosner [13]. They developed synthesis algorithms for finite-state discrete systems, and showed that finding a winning strategy for the game was equivalent to synthesizing a controller that satisfied the requirements.

Recently, on-the-fly algorithms have been proposed to solve the issue of controller synthesis for discrete and dense-time systems, method restricted to finite-state systems [15]. The 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. In comparison, our general method can be applied as such, to both infinite and finite systems. In both cases, the synthesis relies on the same proof theory.

Asarin et al. also apply game techniques to construct discrete controllers, and the system is modeled by a timed automaton with trivial continuous dynamics [1]. The authors develop fixpoint algorithms in order to compute the maximal strategy. The method uses a "predecessor" operator that might imply a resource-consuming implementation, and also the exploration of possibly unreachable states. Similar algorithms suited for model-checking are proposed by Maler, Pnueli and Sifakis, who give a simple solution to the problem, without generating lengthy automata trees [11].

In the rest of the paper, we introduce our method for solving the synthesis task, and we illustrate it on a producer-consumer system.

2 Example: A Producer - Consumer Application

Let us assume that we are given the task of designing a controller for a *First-In-First-Out* (FIFO) memory buffer (or stack) to which a specific *producer* process adds data, while a particular *consumer* takes away data from the buffer, yet respecting some predefined constraints. This kind of pipelined controller could be useful, for instance, in the design of certain hardware devices.

Our goal is to ensure that the producer can always provide at least one new

input to the buffer, that is, the buffer is never full after the consumer has finished its round. We choose to show our proposed methodology on a *parameterized* model, where the parameter is the capacity of the buffer.

In the example that we present, we suppose that the producer places items at one end of the buffer, and the consumer removes items at the other end (Figure 1 a)). However, this is just a modeling point of view, since the methodology applies also if they operate at the same end of the buffer (Figure 1 b)).

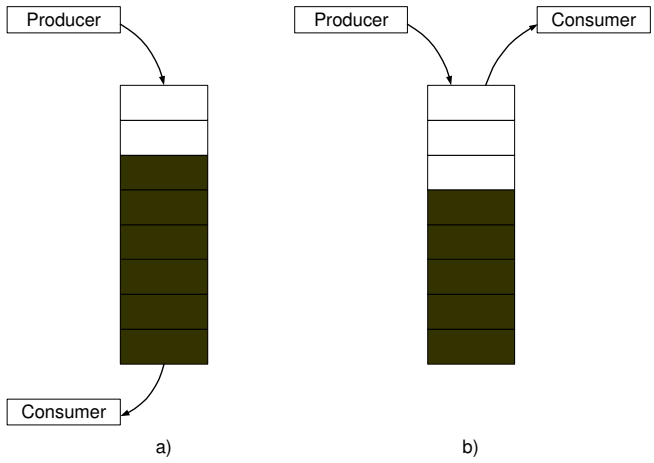


Figure 1: The producer-consumer example: a) FIFO, b) Stack (LIFO)

The first step is to model the system. We start by imagining a game between the controller, represented by the controllable variables, and the plant, modeled by the uncontrollable ones. The players take turns and make moves with respect to given rules. For instance, each time the system executes, the controller has to change the input, by adding at most two items at a time, into the virtual buffer. Hence, as a first constraint, we impose the fact that, at each round, the controller is compelled to throw "data" into the buffer (that is, add one or two items), thus it can not skip. Similarly, the disturbance, or the plant, may choose to remove at most two items at a time, or leave the system state unchanged. Another constraint, this time for the plant, is the fact that the latter is not allowed to skip unless it has removed one item from the buffer, in the immediate previous step. Also, if the plant did not remove any item in the current round, it has to remove two items, next round. Similarly, after the consumer has removed two items, it is mandatory that it removes only one item next time. Last but not least, if the consumer removes one item in the current round, it can nondeterministically choose to remove one or two items from the buffer, or leave it unchanged. The players move sequentially, and the observer sees the start of each round and the end of it, without noticing

the intermediate states.

The rules of the game are those described above, and the goal of the controller is to find a way to keep some required property true, during the execution of the system. In section 4 we model the mentioned behavior, formally.

The variables that describe the state of the system are as follows:

• C : Nat - models the content of the buffer as updated by the consumer, at the end of each round of the game; it represents the value that is apparent to the external observer;

• r : $\{0,1,2\}$ - represents the quantity removed by the consumer, from the buffer;

• cap : Nat- models the capacity of the buffer, yet not less than 4 locations ($cap \geq 4$), for the buffer to be sufficiently large.

The goal of the producer (controller) is a postcondition formalized as an "always" temporal property:

$$\Box Q = \Box(0 \leq C < cap),$$

that is, the controller loses the game if the consumer manages to leave the buffer full, after its respective update. By enforcing $\Box Q$, we ensure that there is a continuous activity at the producer end of the buffer.

In spite of the partly nondeterministic moves of the consumer (disturbance), the producer should be able to enforce $\Box Q$. Having a way to keep it true during the entire execution of the system is equivalent to synthesizing a controller for invariance.

In this paper, we focus on synthesizing such a controller, within the mentioned setup.

3 Background

In this section, we give an overview of contracts, and introduce action systems as a special kind of contract. 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. We model program statements by contracts. A *contract* S is built according to the syntax below:

$$\langle f \rangle \mid \{p\} \mid [p] \mid S_1 \,;\, S_2 \mid \{x :=\, x' \mid b\} \mid [x :=\, x' \mid b]$$

Here, p ranges over *state predicates* ($\Sigma \rightarrow$ Bool), f over *state transformers* ($\Sigma \rightarrow \Gamma$), and $x :=\, x' | b$ is a *state relation* ($\Sigma \rightarrow \Sigma \rightarrow$ Bool), where Σ 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, $\langle x := e \rangle$ is a special kind of update where the state transformer is expressed as an assignment). We use the name skip for the identity update. The *assertion* $\{p\}$ leaves the state unchanged if p holds and aborts otherwise, whereas the *assumption* $[p]$ also leaves the state unchanged if p holds, but terminates miraculously otherwise. In the *sequential composition* $S_1 ; S_2$, contract S_1 is first carried out, followed by S_2.

The *angelic nondeterministic assignment* or *angelic update*, $\{x := x' \mid b\}$, lets the angel choose the final state, among those that satisfy the boolean condition b, whereas in the *demonic nondeterministic assignment* (*demonic update*), $[x := x' \mid b]$, the choice is demonic. 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 (i.e, it establishes any postcondition, even *false*). A sequence of an angelic and a demonic update is interpreted as a *game* with the angel and the demon as players.

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 σ from which S is guaranteed to end in a state of q. Thus, $S.q$ is the *weakest precondition* of S to establish postcondition q. The intuitive description of contract statements can be used to justify the following definition of the weakest precondition semantics:

$$(S_1 ; S_2).q = S_1.(S_2.q) \quad (1)$$

$$\{x := x' \mid b\}.q = (\exists x' \bullet b \wedge q[x := x']) \quad (2)$$

$$[x := x' \mid b].q = (\forall x' \bullet b \Rightarrow q[x := x']) \quad (3)$$

These definitions are consistent with Dijkstra's original semantics for the language of guarded commands [9], 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 $(\mu X \bullet S)$ or $(\nu X \bullet S)$, depending on whether contract X can be invoked a finite number of times, or infinitely, respectively. An important particular case of recursion is the $do - od$ loop, which is defined in the usual way: $\mathsf{do}\ g \rightarrow S_1\ \mathsf{od} \triangleq (\mu X \bullet \mathsf{if}\ g\ \mathsf{then}\ S_1 ; X\ \mathsf{else\ skip\ fi})$.

In this paper, we consider the special case of an *action system*, as a contract of the form

$$Sys(y) \triangleq \mathsf{begin\ var}\ x \bullet\ S_0 ;\ \mathsf{do}\ g \rightarrow S_1\ \mathsf{od\ end} \quad (4)$$

Here, Sys contains an *initialization* statement S_0 and the action statement S, which has a guarded form, $S = g \rightarrow S_1$, where g is a boolean condition called the

guard, and S_1 is the *body* of S. The initialization statement typically introduces some local variables, x, for the action system, and initializes 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 the action body S_1 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. In consequence, $p\ \{\!|S|\!\}\ q\ \equiv\ p \subseteq S.q$ denotes the total correctness of statement S with respect to precondition p and postcondition q.

We say that *contract S is refined by contract S'* (written $S \sqsubseteq S'$), if S' preserves all the correctness properties of S, which is equivalent to $S \sqsubseteq S' \equiv (\forall q \bullet S.\, q \subseteq S'.\, q)$.

A refinement rule is an inference rule that allows us to deduce that a certain refinement $S \sqsubseteq S'$ is valid. Adding choices to an angelic update and removing choices from a demonic update are both valid refinements. Equality "$=$" of contracts can be used as refinement.

We will use the rule of *propagating an assertion backwards, into an angelic nondeterministic assignment*, which is given below:

$$\{x :\!= x' \mid b\}\ ;\{q\} = \{x :\!= x'\ |b \wedge q[x :\!= x']\} \tag{5}$$

4 The Producer-Consumer Model as an Action System

The process of controller synthesis is gradual, since it starts with a nondeterministic model of the system, which has to be further adjusted correctly, in order to be brought closer to the implementable level. This justifies our decision to specify the actions of the producer, as an angelic nondeterministic assignment. Thus, the controller behavior is described as follows:

$$Prod = \{C :\!= C'\ |C < C' \leq C + 2\} \tag{6}$$

The boolean condition of the assignment ensures that the producer adds one or two items to the buffer. Should we not require this condition to hold, the basic angelic behavior is not enforced.

As the consumer is partly uncontrollable, it behaves demonically. In conse-

quence, it is modeled by a demonic nondeterministic assignment:

$$\begin{aligned} Cons &= [r, C :\!= r', C' \,|(r = 0 \Rightarrow r' = 2) \wedge \\ &\qquad (r = 1 \Rightarrow r' \in \{0, 1, 2\}) \wedge \\ &\qquad (r = 2 \Rightarrow r' = 1) \wedge \\ &\qquad C' = C - r'] \\ &= [r :\!= r' \,|(r = 0 \Rightarrow r' = 2) \wedge \\ &\qquad (r = 1 \Rightarrow r' \in \{0, 1, 2\}) \wedge \\ &\qquad (r = 2 \Rightarrow r' = 1)] \,; \\ &\qquad C :\!= C - r \end{aligned} \tag{7}$$

The contract $Cons$ regulates the moves of the consumer, as mentioned in section 2.

The producer is responsible to enforce the safety property $\Box Q$, formalized previously, that is, at each turn, it should choose an appropriate number of items to add to the buffer, such that the latter can never be left fully occupied, by the consumer. The property should be guaranteed, regardless of the demonic nondeterministic moves.

Further, we model the producer and the consumer, together, as the action system below, where we substitute relation (6) for $Prod$, and (7) for $Cons$. The system terminates upon the completion of the process. This is decided by an external device, modeled by contract Dev. However, we choose here to model a non-terminating loop. At will, the guard $true$ can be replaced by a non-trivial one.

$$\begin{aligned} &Buffer(r, C, cap : \mathsf{Nat}) = \\ &\quad \mathsf{begin} \\ &\quad\; r := 1 \,;\, C := 4 \,;\, cap := 8; \\ &\qquad \mathsf{do}\; true \;\rightarrow Prod \,;\, Cons \,;\, Dev\; \mathsf{od} \\ &\quad \mathsf{end} \end{aligned} \tag{8}$$

The discrete controller (that is, the producer) of the buffer will result out of certain transformations of the nondeterministic behavior of the angel, given by (6), into a more deterministic one, such that the safety property $\Box Q$ is enforced. During the process, the demonic behavior stays unchanged.

5 Synthesis of Logic Controllers for Discrete Systems

As already mentioned, the process of controller synthesis can be seen as a game between two players, the controller and some disturbance. We assume that the be-

havior of the disturbance is hostile, thus we would like the controller to guarantee the requirements despite the action of the disturbance. Therefore, the controller is *the angel*, and the disturbance or plant is *the demon*. During the game, the goal of the angel is to force the system to remain inside a certain "good" subset of the state space, whereas the demon's goal is to force the system to leave this same subset.

In our approach, the *discrete system* is modeled by an action system given by (4). We define the action of the loop, as $S \triangleq g \rightarrow A;D$. Here, A contains angelic choices and D demonic ones. The values of the variables are chosen either by the controller or by the plant. The contract A that models the *controller* is, in our case, an angelic nondeterministic assignment of the form $\{x :=\ x' \mid cd\}$, which should be further transformed into an implementable construct. The *plant* is modeled by statement D, which describes the demonic behavior. The *requirement* is encoded as a safety property, expressed as a subset of the state space.

The goal of the controller is to continually observe the plant, and force control events at appropriate times, such that the plant always remains within the safe set of states.

Given the action system (4), and assuming that at each round of the game, sequential angelic and demonic choices determine the next state of the game, we can intuitively split the synthesis problem into two subproblems:

(a) Enforcing the safety (or liveness) property, equivalent to calculating the largest set of initial states from which the angel can always win with respect to that property;

(b) In case this set exists, constructing a controller that renders it, equivalent to extracting a winning strategy, or a set of winning strategies, for the angel.

5.1 Enforcing the Required Property

Designing a controller for invariance implies that we specify some safety property that should be enforceable by the angel during system execution. Here, we express this property as an "*always*" ($\Box$) temporal property.

In the following, we show how we can compute the precondition for the angel to enforce the property $\Box q$ in the action system Sys, given by (4), where the contract S is of the form $S = g \rightarrow A\ ;\ D$. Applying the result proved in [6], to our case, we get the following:

$$\begin{array}{ll} & p\ \{\!|\mathsf{do}\ g \rightarrow A\ ;\ D\ \mathsf{od}|\!\}\ \Box q \\ \equiv & p \subseteq (\nu X \bullet \{q\}\ ;\ [g]\ ;\ A\ ;\ D\ ;\ X).false \end{array} \tag{9}$$

where $\neg g$ is the exit condition, which is tested before entering the loop.

Formula (9) shows that we can reduce the question of whether a temporal property 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.

If we want to synthesize controllers for reachability, the angel has to guarantee liveness properties, modeled as "*eventually*" ($\diamondsuit$) properties. Here, we only present the correctness rule for guaranteeing a special liveness property, of the form $\Box(p \Rightarrow \diamondsuit q)$, p, q predicates. This property is called *weak liveness* or *weak response*. The 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, or if the angel is at least able to keep the system into a state of $\neg p$, forever. Thus, we define the contract WLiv.p.q:

$$\begin{array}{ll} & \mathsf{WLiv.p.q} \\ \triangleq & (\nu X \bullet [\neg p]\,;[g]\,;A\,;D\,;X \sqcap [p]\,;(\mu Y \bullet [\neg q]\,;\{g\}\,;A\,;D\,;Y); \\ & [g]\,;A\,;D\,;X) \end{array}$$

Since $\forall \sigma \bullet \sigma \,\{\!|\mathsf{do}\ g \rightarrow A\,;D\ \mathsf{od}|\!\}\, \Box(p \Rightarrow \diamondsuit q) \equiv (\mathsf{WLiv.p.q}).false.\sigma$, we have further that

$$\begin{array}{ll} & p_0 \,\{\!|\mathsf{do}\ g \rightarrow A\,;D\ \mathsf{od}|\!\}\, \Box(p \Rightarrow \diamondsuit q) \\ \equiv & p_0 \subseteq (\nu X \bullet [\neg p]\,;[g]\,;A\,;D\,;X \sqcap [p]\,;(\mu Y \bullet [\neg q]\,;\{g\}\,;A\,;D\,;Y); \\ & [g]\,;A\,;D\,;X).false \end{array}$$

Assuming the action system in a recursive form, Back and von Wright show, in [6], how to prove enforcement of temporal properties by using usual invariant-based methods, rather than the more costly fixpoint computation algorithms. Thus, in order to make the proof of a safety property practical, we adapt their result to our case. Moreover, we introduce a new inference rule, useful for enforcing weak response properties. Both rules are shown in Lemma 1.

Lemma 1 *Assume the following action system*:

$$Sys(y) \;=\; \mathsf{begin\ var}\ x \bullet\ S_0\,;\ \mathsf{do}\, g \rightarrow A\,;D\ \mathsf{od\ end}$$

Then:

(a) *Always-properties can be proved using invariants*:

$$\frac{p \subseteq I \qquad g \cap I \,\{\!|\, A\,;D \,|\!\}\, I \qquad I \subseteq q}{p \,\{\!|\, \mathsf{do}\, g \rightarrow A\,;D\ \mathsf{od} \,|\!\}\, \Box\, q}$$

where p, q are predicates.

(b) *Weak liveness-properties can be proved using invariants and termination arguments, as follows*:

$$\frac{p_0 \subseteq I \cup r \qquad g \cap r \,\{\!|\, A\,;D \,|\!\}\, I \cup r \qquad p \cap r \subseteq q \cup I \qquad pre \,\{\!|A\,;D|\!\}\, post}{p_0 \,\{\!|\, \mathsf{do}\, g \rightarrow A\,;D\ \mathsf{od} \,|\!\}\, \Box(p \Rightarrow \diamondsuit q)}$$

where p, r are predicates, $pre \triangleq \neg q \cap g \cap I \cap t = w$, $post \triangleq (q \cap (I \cup r)) \cup (I \cap t < w)$, and the state function t ranges over some well-founded set. □

The first rule states that proving the *always* property for the loop of the action system Sys, with the precondition of the loop established by the initialization, is in fact equivalent to showing that a predicate $I \subseteq q$ is an invariant of Sys. Therefore, proving the safety property q by proving an invariance property subsumes the following obligations:

1. Invent a predicate I, such that $I \subseteq q$ holds.

2. Prove that I is established by the initialization S_0, that is, $p \subseteq I$, where p is the predicate that holds after S_0.

3. Prove that I is preserved by the action $g \rightarrow A\,;D$, that is, $g \cap I \subseteq A.(D.I)$.

It then follows that, if the above conditions hold, the angel has a winning strategy, A, thus a controller for invariance can be synthesized.

The second rule of Lemma 1 shows the proof obligations that we have when carrying out controller synthesis for weak liveness. The predicate r includes the states of p that have already been followed by a state of q. Therefore, the angel is considered to have a winning strategy if it can find a way to keep the system in I, trying to decrease t, or wander within r.

5.2 Extracting the Control Strategy

After having established that the angel can enforce a certain 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 Sys, as defined in Lemma 1, we know that the contract

$$S = \{I\}\,;A\,;\{D.I\}\,;D\,;\{I\}$$

can replace $A\,;D$ inside the body of the loop, since S preserves the invariant, trivially. In consequence, we can rewrite A by using the information supplied by $\{D.I\}$, such that we force the angel to restrict its choices only to the ones that establish I.

In our case, $A = \{x :\!= x' \mid cd\}$, thus, we can assert $D.I$ after A, such that we get the contract $A\,;\{D.I\}$. Next, we use this assertion to further refine the new contract, by propagating $\{D.I\}$ backwards. In this way, we strengthen the boolean condition inside the angelic nondeterministic assignment. As a result, the

angelic choices are restricted according to the propagated information. We apply the refinement rule (5), as follows:

$$\begin{array}{ll} & \{x :=_{=} x' \mid cd\} \,;\, \{D.I\} \\ = & \\ & \{x :=_{=} x' \mid cd \wedge D.I[x :=_{=} x']\} \end{array} \tag{10}$$

In principle, this is a transformation that does not actually favor our agent, it rather makes the demon happy, since it decreases the set of final states that the angel can choose from. The demon can still achieve its goals, while the angel's choice possibilities are being removed. However, the refinement in the specified context makes the behavior of the angel more predictable. Moreover, the transformation given by (10) preserves the invariant I proved by means of Lemma 1.

Further, we might need to refine $A' = \{x :=_{=} x' \mid cd \wedge D.I[x :=_{=} x']\}$ into a program. For this, we apply suitable refinement rules [4], which guarantee that the implementation preserves the correctness of the model.

6 Applying the Synthesis Method

We return to our case-study, and start the controller synthesis by checking whether the safety property $\Box Q$, given in section 2, as $\Box(0 \leq C < cap)$, can be enforced by the producer. In case this is possible, we move along the line established above, to extract the control strategy.

Given the system model as the action system $Buffer$ defined by (8), the steps that we take are as follows:

• A1) Firstly, we find a predicate $I \subseteq Q$. Thus, we choose I as follows:

$$\begin{array}{rcl} I & = & (r = 0 \wedge 0 \leq C < cap) \vee \\ & & (r \neq 0 \wedge 0 \leq C \leq cap - 2) \end{array}$$

Proving that $I \subseteq Q$ is straightforward.

• A2) Next, I has to be an invariant of the action system $Buffer$. Note that the contract Dev preserves the invariant ($I\{\!|Dev|\!\}I$ holds), since it does not interfere with the variables mentioned in I. The invariant is trivially established by the initialization statement:

$$\begin{array}{ll} & p \\ = & \\ & r = 1 \wedge C = 4 \wedge cap = 8 \\ \subseteq & \\ & I \end{array}$$

$$
\begin{array}{ll}
 & Prod.(Cons.I) \\
\equiv & \{\textbf{substitute contract Cons}\} \\
 & Prod.([r :\!= r' \mid \\
 & \quad (r = 0 \Rightarrow r' = 2) \wedge \\
 & \quad (r = 1 \Rightarrow r' \in \{0,1,2\}) \wedge \\
 & \quad (r = 2 \Rightarrow r' = 1)] \,;\, C = C - r).I \\
\equiv & \{\textbf{rules } (1),\ (3)\} \\
 & Prod.(\forall r' \bullet \\
 & \quad ((r = 0 \Rightarrow r' = 2) \wedge \\
 & \quad (r = 1 \Rightarrow r' \in \{0,1,2\}) \wedge \\
 & \quad (r = 2 \Rightarrow r' = 1)) \\
 & \Rightarrow (I[C :\!= C - r])[r :\!= r'] \\
\equiv & \{\textbf{substitute contract Prod},\ \textbf{simplify}\} \\
 & \{C :\!= C' \mid C < C' \leq C + 2\}. \\
 & \quad ((r = 0 \Rightarrow 2 \leq C \leq cap) \wedge \\
 & \quad (r = 1 \Rightarrow 2 \leq C \leq cap - 1) \wedge \\
 & \quad (r = 2 \Rightarrow 1 \leq C \leq cap - 1)) \\
\equiv & \{\textbf{rule } (2)\} \\
 & (\exists\, C' \bullet C < C' \leq C + 2 \wedge \\
 & \quad ((r = 0 \Rightarrow 2 \leq C' \leq cap) \wedge \\
 & \quad (r = 1 \Rightarrow 2 \leq C' \leq cap - 1) \wedge \\
 & \quad (r = 2 \Rightarrow 1 \leq C' \leq cap - 1))) \\
\supseteq & \\
 & ((r = 0 \wedge 0 \leq C < cap) \vee \\
 & (r \neq 0 \wedge 0 \leq C \leq cap - 2))
\end{array}
$$

Figure 2: Proof of the invariant

Then, we prove that I is preserved by the action of the loop, that is,

$$I \subseteq Prod.(Cons.(Dev.I)) \Leftarrow I \subseteq Prod.(Cons.I).$$

The proof is shown in Figure 2. We have also proved the invariant in the *Prototype Verification System (PVS)* [8]. In consequence, irrespective of the chosen value of r, the producer has a way of enforcing $\Box Q$, hence to keep the buffer not fully filled, at the end of each round of the game.

• B) In the following, we apply rule (10), to refine the contract $Prod$, given by (6), by propagating backwards the assertion

$$
\begin{array}{ll}
 & \{Cons.I\} \\
= & \\
 & \{(r = 0 \Rightarrow 2 \leq C \leq cap) \wedge \\
 & (r = 1 \Rightarrow 2 \leq C \leq cap - 1) \wedge \\
 & (r = 2 \Rightarrow 1 \leq C \leq cap - 1)\},
\end{array}
$$

such that all the possible choices, except for the ones that establish I, are removed.

Below, we show the derivation that leads to the control strategy of the producer, where any of its choices satisfies I:

$$
\begin{array}{ll}
 & \{C :\!= C' \mid C < C' \leq C + 2\} \; ; \{Cons.I\} \\
= & \{\textbf{rule } (10)\} \\
 & \{C :\!= C' \mid C < C' \leq C + 2 \wedge \\
 & \quad (r = 0 \Rightarrow 2 \leq C' \leq cap) \wedge \\
 & \quad (r = 1 \Rightarrow 2 \leq C' \leq cap - 1) \wedge \\
 & \quad (r = 2 \Rightarrow 1 \leq C' \leq cap - 1)\} \\
= & \\
 & Prod_f
\end{array}
\tag{11}
$$

The contract $Prod_f$, given by (11), represents the winning strategy of the producer to always keep $Q \equiv true$, during execution. Concretely, if one replaces C' with $C + 1$, or $C + 2$, he/she knows exactly how to move next, that is, to add one or two items to the buffer, depending on the value of C, previously updated by the consumer. The strategy ensures a win for the angel, for whatever choices selected by the demon.

Thus, by strengthening the boolean condition of the angelic nondeterministic assignment $Prod$, given by (6), through the information that we have got by propagating backwards the weakest precondition of contract $Cons$ with respect to postcondition I, we have eliminated the angelic choices that would not establish I, such that, in its new form, the producer can blindly select its moves, yet satisfying $\Box Q$, which has been our design target. Now, we can safely replace $Prod$ by $Prod_f$ in the action system $Buffer$.

An interesting extension of the analyzed example is to try to keep the content of the buffer within certain specified limits. In this case, additional information, which describes the conditions at the other end of the buffer, should be also formalized and propagated. Extending the ideas introduced here to a more general producer-consumer problem would, indeed, lead to the construction of a correct and reliable template for such a class of systems, where not only the capacity, but also the number of inputs and outputs, that is, the choices of the producer and the consumer, respectively, are parameterized. Both issues are subjects of further studies.

7 Conclusions

In this paper, we have tackled the problem of discrete controller synthesis, by modeling the system as an action system, and the synthesis process as a *two-player* game. The players are the controller, called the *angel*, and the plant, called

the *demon*, which make moves sequentially, according to some contract statement. The goal of the angel is a safety temporal property. The angel-demon game formalization in the *weakest precondition* framework was introduced by Back and von Wright [3].

In general, relationships between agents may involve both cooperation and competition. To make the synthesis possible, in our case, the angel competes with the demon.

We have started with an angelic nondeterministic assignment as the model of the controller, and a demonic update for the behavior of the plant. The synthesis subsumes two main steps. Firstly, we check whether the angel can enforce the required behavior (A1, A2 of section 6 show how the first step is applied in practice). We use a certain inference rule that reduces proving safety properties to invariance proofs. If this first step holds, we move toward extracting the safe set of strategies, or, sometimes, toward implementing a specific control strategy (step B in section 6).

In order to restrict the angelic choices to the ones that establish the safety property, we have propagated backwards the assertion of the weakest precondition of the demon, to establish the invariant, through the angelic nondeterministic assignment. This method provides us with means of rewriting the angelic nondeterministic assignment, by using the information that we obtain from the fact that the required safety property is enforced on the initial model. Hence, we replace the initial angelic update by a new contract that refines the former, in this context. The end-result is a correct-by-construction controller, tailored to the required behavior.

We believe that our method is particularly useful when the discrete system is as much constrained as it is nondeterministic. Moreover, the technique proved well suited for the situations when the game lasts more than one round, that is, neither the demon nor the angel have a one move strategy to win the game.

An illustrative case-study has shown the application of the proposed approach, in practice. Due to Lemma 1 and the method described in section 5.2, we have synthesized an invariance controller for a *producer - consumer* - like system.

Distinctly from the fixpoint symbolic synthesis algorithms proposed in [1, 10, 11], our games-based method is fit for interactive theorem proving (PVS [8], HOL etc.). To support this claim, we have proved the invariance property of the producer-consumer system, in PVS. Thus, our approach works for models with unbounded variables, too.

Future research targets the development of games-based synthesis techniques for real-time control systems, within the action systems framework.

Acknowledgments. The authors thank Viorel Preoteasa and Tiberiu Seceleanu for their comments on this paper.

References

[1] R. Asarin, O. Maler, and A. Pnueli. "Symbolic controller synthesis for discrete and timed systems". In P. Antsaklis, W.Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems II*, volume 999, *Lecture Notes in Computer Science*, Springer-Verlag, 1995.

[2] R. J. R. Back and K. Sere. "Stepwise refinement of action systems". *Structured Programming*, 12:17-30, 1991.

[3] R. J. R. Back and J. von Wright. "Games and winning strategies". *Information Processing Letters*, 53(3):165-172, 1995.

[4] R. J. R. Back and J. von Wright. *Refinement Calculus: A Systematic Introduction*. Springer-Verlag, 1998.

[5] R. J. R. Back and J. von Wright. "Contracts, games and refinement". *Information and Computation*, 156:25-45, 2000.

[6] R. J. R. Back and J. von Wright. "Enforcing behavior with contracts". *Technical Report* nr. 373, TUCS, 2000.

[7] R. J. R. Back and J. von Wright. "Verification and refinement of action contracts". *Technical Report* nr. 374, TUCS, 2000.

[8] J. Crow, S. Owre, J. Rushby, N. Shankar, and M. Srivas. "A tutorial introduction to PVS". In *WIFT'95 Workshop on Industrial-Strength Formal Specification Techniques*, April 1995.

[9] E. W. Dijkstra. A discipline of programming. *Prentice-Hall International*, 1976.

[10] G. Hoffmann and H. Wong-Toi. "Symbolic synthesis of supervisory controllers". In *Proceedings of the American Control Conference*, Chicago, IL, pages 2789-2793, June 1992.

[11] O. Maler, A. Pnueli, and J. Sifakis. "On the synthesis of discrete controllers for timed systems". In *Proceedings of STACS'95, E. W. Mayr and C. Puech (Eds.)*, volume 900 of *Lecture Notes in Computer Science*, 229-242, Springer-Verlag, 1995.

[12] C. Morgan. *Programming from Specifications*. Prentice-Hall International, 1998.

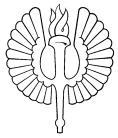
[13] A. Pnueli and R. Rosner. “On the synthesis of a reactive module”. In *Proceedings of the 16th ACM Symposium on Principles of Programming Languages*. 179-190, 1989.

[14] P.J. Ramadge and W.M. Wonham. “Supervisory control of a class of discrete event processes”. *SIAM Journal of Control and Optimization 25* 206-230, 1987.

[15] S. Tripakis and K. Altisen. “On-the-fly controller synthesis for discrete and dense-time systems”. In *World Congress on Formal Methods, FM’99*, 1999.

Turku Centre for Computer Science
Lemminkäisenkatu 14
FIN-20520 Turku
Finland

http://www.tucs.fi



University of Turku
- **Department of Information Technology**
- **Department of Mathematics**

Åbo Akademi University
- **Department of Computer Science**
- **Institute for Advanced Management Systems Research**

Turku School of Economics and Business Administration
- **Institute of Information Systems Science**