Symbolic Synthesis of Finite-State Controllers for Request-Response Specifications

Nico Wallmeier, Patrick Hütten, and Wolfgang Thomas

Lehrstuhl für Informatik VII
RWTH Aachen, 52056 Aachen, Germany
{wallmeier, phuetten, thomas}@i7.informatik.rwth-aachen.de

Abstract. We present a method to solve certain infinite games over finite state spaces and apply this for the automatic synthesis of finite-state controllers. A lift-controller problem serves as an example for which the implementation of our algorithm has been tested. The specifications consist of safety conditions and so-called “request-response-conditions” (which have the form “after visiting a state of P later a state of R is visited”). Many real-life problems can be modeled in this framework. We sketch the theoretical solution which synthesizes a finite-state controller for satisfiable specifications. The core of the implementation is a convenient input language (based on enriched Boolean logic) and a realization of the abstract algorithms with OBDD's (ordered binary decision diagrams).

1 Introduction

Automatic verification (model-checking) is one of the most successful contributions of theoretical computer science to system development in industrial applications. A first central point of the model-checking methodology is provided by automata theoretic algorithms in connection with suitably chosen specification logics (usually temporal logics like CTL or LTL). A second prerequisite of efficient model-checking is the “symbolic approach”, namely the description of large state spaces and corresponding transition relations by Boolean formulas and their transformation into OBDD's (ordered binary decision diagrams). Both techniques together have led to a powerful framework which supports large-scale system construction [1].

In the world of verification, the system to be implemented is assumed to be given and has to be “checked”. The automata theoretic methodology, however, provides theoretical algorithms for a much more ambitious task, namely for the automatic synthesis of control programs. Here a state-based system is modeled in a more structured way than by a simple transition graph: In order to capture the interactions between the control program and its environment, the global system is viewed as an arena for a two-person game, i.e. a transition graph where each state is associated to one of the two players (program = player 0, environment = player 1); and in a state associated to player i this player i is allowed to pick a transition to a successor state. Infinite system runs then correspond to infinite plays over this game graph (i.e. infinite paths through the graph); and the specification is completed by a winning condition on such plays which player 0 should try to fulfill. A solution of an infinite game over a finite arena requires to determine those states from which player 0 has a winning strategy, and in this case to construct such a strategy.

There is by now a large reservoir of algorithms and theoretical results on infinite games (see [6], [7], [4] for introductions). Only few algorithms have been implemented, however. For example, in [5] an implementation is presented for the algorithm [9] which solves parity games, motivated by the still open question whether parity games over finite graphs can be solved in polynomial time. Such an implementation is meant to support experiments in order to better understand an open theoretical question rather than to synthesize practical control programs. Synthesis tools for "real" control problems in the game-theoretic framework seem to be missing so far. This is due to three reasons:

1. lack of a powerful input language which allows to specify nontrivial game graphs (using Boolean formulas),
2. the conjunctive format of practical winning conditions,
3. the high complexity of the available synthesis algorithms.

Let us comment on the second and third aspect (which are connected). The classical automata theoretic winning conditions are the Büchi-, the Muller-, and the parity condition. The first simply requires that a certain set of states is visited again and again, the second and the third have the form of disjunctions where an atomic condition says "the set of infinitely often visited states is F", respectively "the maximal color visited infinitely often is the (even) number i" (assuming that a coloring of states by numbers is given). Only the solution of games with a Büchi winning condition can (so far) be done in polynomial time. Moreover, a practical condition usually is a conjunction in which several simple conditions are collected. Its conversion into a disjunction (e.g. a Muller condition) and a subsequent construction of a winning strategy is too costly to allow nontrivial applications.

In the present paper, we describe a method and an implementation which provides progress regarding the first two items above and which gives some insight into the third. We proceed in three steps:

First, a specification language is provided for parameterized definitions of game graphs (using an enriched Boolean logic with Boolean quantifiers, indexed variables and support for binary encoding).

The winning conditions are supposed to be conjunctions of "safety conditions" and "request-response conditions". Here a *safety condition* is represented by a restriction on the states to be visited or transitions to be taken. A *request-response condition* (sometimes also called "assume-guarantee-condition" in the literature) refers to two state-sets or state-properties P, R and has the form

"whenever a state in P is visited, then now or later a state in R is visited"

It is a basic form of liveness condition. By an *RR-game* we mean a game where the winning condition is a conjunction of safety- and request-response-conditions. A large number of practical specifications for reactive systems can be coded in this format of RR-games.

Once the RR-game is specified, the second step consists in reducing it to a simple Büchi game (which will involve a problematic blow-up of the number of states). In a third step, the resulting Büchi game is solved. From this solution a finite-state controller for the original RR-game is extracted, respectively an error scenario is provided in terms of a strategy for the environment which falsifies controller's winning condition.

We validate this approach by showing that one can solve small but concrete control problems which are coded as RR-games. As an example we provide an analysis and solution of a lift controller problem which is well below industrial dimensions but nevertheless too complicated to immediately suggest a commonsense solution. Not surprisingly, the complexity of implementing and evaluating conjunctions of request-response-conditions remains the main problem for the treatment of larger examples.

The paper is structured as follows: A short first section describes the example controller problem. The theoretical analysis is done in the subsequent two sections: first on the level of abstract states, secondly on the symbolic level (working with Boolean formulas). Then we give comments on our implementation, its application in the presented example, and some consequences to be drawn.

Our approach should be compared with recent work of Melcher [3] who emphasizes the modular construction of winning strategies. However, the winning conditions in [3] are only conjunctions of safety conditions and a *single* fairness condition of the form "if P_1 is visited infinitely often, then so is P_2".

The task of handling long conjunctions of request-response conditions (and of fairness conditions) remains to be studied in more detail. The conclusion gives some hints on our current efforts in this context.

2 A Lift-Controller Problem

We consider a lift controller for two lifts in a building with e floors. There are two players: the lift controller, which has to navigate both lifts, and the environment, represented by persons which request the lifts, respectively the target floors. It is convenient (but not essential) to assume that the two players act in alternating moves. Thus in a turn of the controller each of the two lifts is directed to one of the e floors. In a turn of the environment the persons on the floors and in the two lifts can enter new requests. We distinguish between requests on the floors (that a lift should come) and requests in the lifts (to which floor to move). The first one can be fulfilled by either of the lifts whereas the second one only by the lift in which the request is issued.

Here are the requirements which the lift controller should to satisfy. This list of conditions is:

1. After a floor is requested a lift will eventually come to this floor. (If a request is issued inside a lift, this lift has to move to the destination.)
2. The highest and the ground floor are served directly ("express floors").
3. In the second floor is the post office: We assume a lift stopping there waits one extra move before proceeding (loading and unloading of mail).
4. The two lifts never stop at the same time in the second floor.
5. No lift skips a requested floor on its way.
6. A lift moves to an unrequested floor only if there is no open request.

The next two conditions are assumptions of the environment's behaviour; they are added here to give the controller better chances to win:

7. At most one person enters a lift at a time (so that the new desired destination is unique).
8. At each moment, there are at least three floors which are not requested.

Some of the conditions above may seem artificial or very restrictive; for example, condition 2 is included in order to simplify the technical analysis for purpose of exposition. The results on satisfiability of the requirements would not be changed if condition 2 would just ask that only one of the "express floors" is to be served immediately.

We would like to answer the following question: For which values of e can the lift controller fulfill all these requirements? Intuitively, increasing e will make it harder to do this. For the cases where the requirements can be met, an appropriate control program (winning strategy of the corresponding infinite game) should be provided.

3 Theoretical Analysis over Abstract States

3.1 RR-Games

We apply the terminology of infinite two-person games as presented in [6], [4]. Player 0 acts as the controller and player 1 as the environment (the collection of people using the lift). The game is specified by a game graph and a winning condition for player 0. A game graph is of the form $G = (Q, E)$ where $Q = Q_0 \dot\cup Q_1$ is the finite set of states (Q_0 the ones of the controller and Q_1 for the environment) and $E \subseteq Q \times Q$ is the transition relation. A sink state s is always included allowing only a transition back to s. A play of a game is an infinite path in the game graph, i.e. an ω-word $\rho = q_0 q_1 \dots$ over the states with $(q_i, q_{i+1}) \in E$.

The winning condition specifies when player 0 wins a game ρ. (A play reaching the designated sink state s and hence staying in this state is assumed to be won by player 1.) If player 0 can enforce to win when starting from state q (so that the play will satisfy the winning condition), we say that player 0 *wins from* q. By W_i we denote the *winning region* of player i, the set of states from which player i wins. As usual, a *strategy* for player i is a function which maps any play prefix ending a state of Q_i to a state which can be visited next. We distinguish positional strategies and finite-state strategies. In the first case, the value only depends on the currently last state of a play prefix, in the second, the function is computable by a finite automaton over words from Q and with output alphabet Q. The effective construction of finite-state strategies is thus a method for synthesizing control programs. For more details see e.g. [6].

We now code the example of Section 2 as a RR-game. The states represent the configurations of the system: Position of the lifts, currently requested floor numbers, and who's turn it is to make the next move. Regarding the winning condition for player "controller", we observe that the first requirement leads to a number of request-response-conditions. We distinguish which floor is requested and how the request arises: Note that this can happen by pressing a button on some floor (lift should fetch a person) or by pressing a button in one of the two lifts (choice of destination). This yields $3e$ individual request-response conditions from requirement 1. With a little reflection we can reduce this number to $3(e-2)$, using the special role of the top and bottom floors (which have to be served directly). We can capture this requirement by cancelling all states and transitions from the game graph which violate this (observe that in this case the liveness condition reduces to a local condition).

All the remaining requirements are safety conditions. These conditions are encoded directly into the game graph by disallowing states and transitions which violate them. For example the last requirement that at least three floors are not requested means to cancel any state which does not satisfy this. The second requirement (ground floor and the highest floor are served directly) is a restriction on the transitions: The controller is allowed only to move the lift to one of these floors once they are requested, or else (when both are requested) moves to the sink state s.

3.2 Reducing RR-Games to Büchi Games

An RR-game can be solved in two stages, first by a reduction to a Büchi game, which in a second step is solved by a well-known algorithm (using positional winning strategies). We follow the approach of game reductions as described in [8]. In the present case, this means to transform a RR-game over a graph $G = (Q, E)$ into a Büchi game over a larger graph $G' = (Q \times S, E')$, where the extra component S is used to simulate the RR-winning condition by a simple Büchi winning condition for a suitably defined set $F \subseteq Q \times S$. (This amounts to a simulation of automata with RR-acceptance condition by Büchi automata.) Formally, we require that each play $\rho = q_0 q_1 \ldots$ over G induces a unique play $\rho' = (q_0, s_0), (q_1, s_1), \ldots$ over G' such that player 0 wins ρ with the RR-condition iff (s)he wins ρ' with the Büchi condition. The construction will ensure the uniqueness of ρ' by a suitable initialization element s_0 and a deterministic updating of the components s_i given the sequence $q_0 q_1 \ldots$.

Theorem 1. *RR-games are reducible to Büchi games, involving a blow-up from n states to $nr2^{r+1}$ states if r request-response conditions are involved.*

Sketch of proof: We consider a game over the graph $G = (Q, E)$ with a winning condition which is a conjunction of conditions "whenever P_i is visited, then now or later R_i is visited" for $i = 1, \ldots r$. If $r = 1$ then we add another pair (P_2, Q_2) with $P_2 = Q_2 = \emptyset$. For the desired reduction we use a marker which indicates which of the r conditions should be fulfilled next (progressing cyclically, and thereby visiting a final state). The expanded game graph $G' = (Q', E')$ and the set F of final states are defined as follows:

- $Q' := Q \times 2^{\{1,\ldots,r\}} \times \{1, \ldots, r\} \times \{0, 1\}$
- $((q, M, m, f), (q', M', m', f')) \in E' \Leftrightarrow$
  - $(q, q') \in E$
  - $M' = (M \cup \{i \mid q' \in P_i\}) \setminus \{i \mid q' \in R_i\}$
  - $m' = \begin{cases} m & \text{, if } m \in M' \\ (m \bmod r) + 1 & \text{, otherwise} \end{cases}$
  - $f' = \begin{cases} 0 \text{ , if } m = m' \\ 1 \text{ , otherwise} \end{cases}$
- $F = Q \times 2^{\{1,\ldots,r\}} \times \{1, \ldots, r\} \times \{1\}$

It can be easily shown that for this Büchi game the above-mentioned claims (of the definition of reduction) are satisfied.

The next result shows that the exponential blow-up in r is unavoidable. We use a family of games where the number of vertices is $O(r)$.

Theorem 2. *There is a family of RR-games $\mathcal{G}^r = (G^r, \Omega^r)$ with $G^r = (V^r, E^r)$ with an initial node v_0 s.t. the following holds:*

- *The number of nodes of G^r is linear in r.*
- *The number of RR-conditions in Ω^r is linear in r.*
- *v_0 is in the winning region W_0 of player 0.*
- *Every strategy automaton of player 0 has at least $2^r \cdot r$ states.*

Sketch of proof: The game graph G^r is given in Fig. 1.

For $1 \leq i \leq r$ there are three different kinds of request-response-conditions:

- $(P_{\underline{i}}, Q_{\underline{i}})$: This condition is requested only in state $\underline{v_i}$ and can only be fulfilled in the states $\overline{v_i}$, $\underline{z_i}$ or e_j with $j \neq i$.
- $(P_{\overline{i}}, Q_{\overline{i}})$: Analogous, with $\underline{v_i}$ for $\overline{v_i}$, $\overline{z_i}$ for $\underline{z_i}$.
- (P_i, Q_i): In connection with $(P_{\underline{i}}, Q_{\underline{i}})$, $(P_{\overline{i}}, Q_{\overline{i}})$ this condition requires that after a P_i-visit and the succeeding visit to w_{r+1} either vertex $\underline{z_i}$ or $\overline{z_i}$ is visited. If the play continues via y_i, the next pair (P_{i+1}, Q_{i+1}) is called.

Q_i^* is an abbreviation for the conjunction of $Q_{\overline{j}}, Q_{\underline{j}}$ with $j \neq i$ and $1 \leq j \leq r$.

The node v_0 is in the winnig region W_0 of player 0 because of the following observations. Here we call a predicate P active if a P-vertex is visited but the corresponding Q-visit still not realized ("P is not fulfilled").

1. By reaching state w_{r+1} not both $P_{\overline{i}}$ and $P_{\underline{i}}$ are active for $1 \leq i \leq r$, because if $P_{\overline{i}}$ is activated in state $v_{\overline{i}}$, $P_{\underline{i}}$ is fulfilled by $Q_{\underline{i}}$ at the same time (and vice versa for $P_{\underline{i}}$).

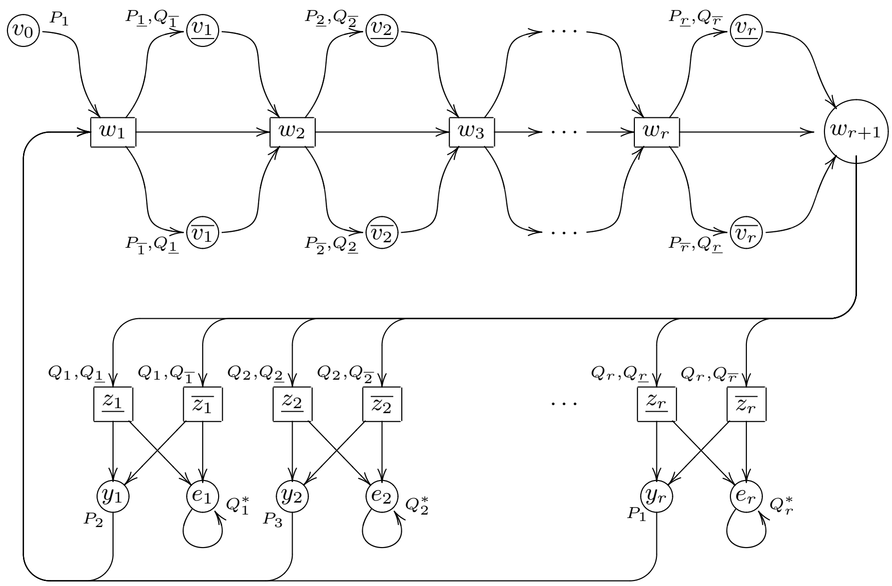

Fig. 1. Game graph G^r

2. While the play does not reach a state e_i, at least one P_j is active when reaching w_1: the fulfillment of a P_i results in the request of $P_{(i \bmod r)+1}$ if player 1 does not move to e_i.
3. Player 0 can only win the game if exactly one P_i is active when reaching w_1. If another P_j with $i \neq j$ is active, player 1 moves to e_k when reaching $\underline{z_k}$ resp. $\overline{z_k}$, whereby at least P_i or P_j can not be fulfilled.

From these observations one can conclude that every strategy automaton needs at least $2^r \cdot r$ states: 2^r for storing whether $P_{\underline{i}}$ is requested (this implies that $P_{\overline{i}}$ is not requested) or not, and r for storing the presently active P_i.

3.3 Solving Büchi Games

Büchi games are solved here along the lines of [6]. We shortly recall the main points, starting with the (simpler) reachability games. In a reachability game, a set F of states is designated, and the winning condition for player 0 requires that some state of F should be visited during the play under consideration.

Remark 3. For reachability games the winning regions W_i are decidable and positional winning strategies are computable (both in polynomial time).

Let us recall the proof: We compute inductively over i the sets

$$Attr_0^i(F) := \{q \in Q \mid \text{from } q \text{ player 0 can reach in} \leq i \text{ moves the set } F\}$$

The inductive construction proceeds as follows:

$$\begin{aligned} Attr_0^0(F) &:= F \\ Attr_0^{i+1}(F) &:= Attr_0^i(F) \ \cup \\ &\quad\ \{q \in Q_0 \mid \exists (q,r) \in E : r \in Attr_0^i(F)\} \ \cup \\ &\quad\ \{q \in Q_1 \mid \forall (q,r) \in E : r \in Attr_0^i(F)\} \end{aligned}$$

Sine the sequence $Attr_0^i(F)$ is increasing and Q is finite, there exists some k where this sequence becomes constant and is the union of all sets $Attr_0^i(F)$. Let $Attr_0(F)$ be $Attr_0^k(F)$ for the first such k. This set consists of the states from which player 0 wins the reachability game. For the (positional) winning strategy for player 0 it suffices to pick, for any Q_0-state in a set $Attr_0^{i+1}(F)$, some transition to a state in $Attr_0^i(F)$ (which exists by construction).

For the solution of Büchi games, we have to determine the states from which player 0 can force infinitely many visits in the given set F.

Remark 4. For Büchi games the winning regions W_i and positional winning strategies are computable (both in polynomial time).

We compute a set $Recur_0(F)$ which contains the F-states from which player 0 can force infinitely many revisits of F. As a preparation we define a modification of the attractor $Attr_0^+(F)$ (with those states from which one revisit of F is possible in at least one move). We let $Attr_0^+(F) = \bigcup\limits_{i \geq 0} A_0^i$ where

$$\begin{aligned} A_0^0 &= \emptyset \\ A_0^1 &= \{q \in Q_0 \mid \exists (q,r) \in E : r \in F\} \cup \{q \in Q_1 \mid \forall (q,r) \in E : r \in F\} \\ i \geq 1 : A_0^{i+1} &= A_0^i \cup \{q \in Q_0 \mid \exists (q,r) \in E : r \in A_0^i\} \cup \\ &\quad\ \{q \in Q_1 \mid \forall (q,r) \in E : r \in (A_0^i \cup F)\} \end{aligned}$$

$Recur_0(F)$ can be defined inductively:

$$\begin{aligned} Recur_0^0(F) &:= F \\ Recur_0^{i+1}(F) &:= F \cap Attr_0^+(Recur_0^i(F)) \\ Recur_0(F) &:= \bigcap_{i \geq 0} Recur_0^i(F) \end{aligned}$$

The winning region of player 0 is now $W_0 := Attr_0(Recur_0(F))$; it is again easy to obtain a corresponding positional winning strategy.

4 Solution on the Symbolic Level

A game graph over the symbolic state space is presented in the form $G = (V, \varphi_0, \varphi_1, \tau)$ where $V = \{v_0, \ldots, v_n\}$ is a finite set of Boolean variables, $\varphi_i(v_0, \ldots, v_n)$ is a Boolean formula for the states of player i, and $\tau(v_0, \ldots, v_n,$

$v'_0, \ldots, v'_n)$ is the transition formula. A concrete state is then an assignment of all variables of V. It can be interpreted as a bit-vector. For the transition formula we use a copy $V' = \{v'_0, \ldots, v'_n\}$ of V. The transition formula is defined over the variables from V and V'. The variables of V describe the source of a transition and V' its target. We assume that $\varphi_0 \wedge \varphi_1 = \mathit{false}$ must hold (every state is owned by only one player).

The computations for solving reachability and Büchi games can be easily transformed to the symbolic level. For a formula λ, set $\mathit{Attr}_0^0(\lambda) := \lambda$ and

$$\begin{aligned}\mathit{Attr}_0^{i+1}(\lambda) := {} & \mathit{Attr}_0^i(\lambda) \;\vee \\ & (\varphi_0 \wedge (\tau \wedge \mathit{Attr}_0^i(\lambda)|_{V\to V'})|_V) \;\vee \\ & (\varphi_1 \wedge \neg(\tau \wedge \neg\mathit{Attr}_0^i(\lambda)|_{V\to V'})|_V)\end{aligned}$$

where '$|_{V\to V'}$' means that the variables are renamed from v to v' for each $v \in V$. The notation '$|_V$' indicates a restriction to V by existential quantification of the remaining variables (of V'). A similar transcripton is possible for the definition of $\mathit{Attr}_0^+(F)$ and $\mathit{Recur}_0(F)$ as defined in the previous section.

The reduction to Büchi games can be transformed to the symbolic state space with the same idea as on the abstract level. For a given RR-game by a game graph $G = (V, \varphi_0, \varphi_1, \tau)$ with $|V| = n$ and set of pairs $(\phi_i,\ \psi_i)$ for $i = 1, \ldots, r$ an expanded game graph $\overline{G} = (\overline{V}, \overline{\varphi_0}, \overline{\varphi_1}, \overline{\tau})$ will be constructed:

- $\overline{V}$ with $(n + r + \lceil \log r \rceil + 1)$ variables:
  - $v_0 \ldots v_{n-1}$ for states of V
  - $v_n \ldots v_{n+r-1}$ to encode the requests (v_{n+i-1} is true if the i-th condition is requested)
  - $v_{n+r} \ldots v_{n+r+\lceil \log r \rceil - 1}$ to encode the next request which should be fulfilled (binary encoding)
  - $v_{n+r+\lceil \log r \rceil}$ flag, if it is a final state
- transition formula $\overline{\tau}$ (here $\mathrm{bin}(x, y, z)$ states that from component y onwards the next z bits are the binary encoding of x):

$$\begin{aligned}\overline{\tau} = \tau \wedge {} & \underbrace{(\bigwedge_{i=0}^{r-1} v'_{n+i} \Leftrightarrow ((v_{n+i} \wedge \neg\psi_i|_{\overline{V}\to\overline{V}'}) \vee \phi_i|_{\overline{V}\to\overline{V}'}))}_{\text{(i)}} \\ & \wedge (\bigvee_{i=0}^{r-1} \mathrm{bin}(i, n+r, \lceil \log r \rceil) \wedge ((\underbrace{v'_{n+i} \wedge \mathrm{bin}'(i, n+r, \lceil \log r \rceil) \wedge \neg v'_{n+2r}}_{\text{(iia)}}) \\ & \vee \underbrace{(\neg v'_{n+i} \wedge \mathrm{bin}'(i+1 \bmod r, n+r, \lceil \log r \rceil) \wedge v'_{n+2r})}_{\text{(iib)}})) \\ & \wedge \underbrace{(\bigvee_{i=0}^{r-1} \mathrm{bin}(i, n+r, \lceil \log r \rceil))}_{\text{(iii)},\alpha} \wedge \underbrace{\alpha|_{\overline{V}\to\overline{V}'}}_{\text{(iv)}}\end{aligned}$$

- $\overline{\varphi_0} = \varphi_0 \wedge \alpha$ and $\overline{\varphi_1} = \varphi_1 \wedge \alpha$
- final state formula $\lambda = v_{n+r+\lceil \log r \rceil}$

The part (i) ensures that the flag that the i-th condition is requested, is only true at the target state of a transition if it was or is now requested and the target state is not a fulfilling assignment of ϕ_i. If the marked condition is satisfied, the end state is a final state and the marker is set to the next condition (iib). Otherwise it is not a final state and the marker does not change (iia). Part (iii) guarantees that only valid values for the marker are possible in a source state of a transition and (iv) is the same for the target state.

5 Back to the Lift Controller

The coding of the lift controller problem as introduced in Section 2 is done with $2\lceil \log e \rceil + 3e + 2$ Boolean variables if e floors are involved:

- $2\lceil \log e \rceil$ variables for the current positions of the lifts (binary encoding)
- e variables for signaling the requests from the floors
- $2e$ variables for the requests from inside the lifts
- one variable to determine the player
- one variable to capture the waiting condition for the post office floor

Due to lack of space we do not present here syntactic details of the input language (in particular, the use of Boolean quantifiers) and on the structure of the user interface. The reader may obtain an impression by inspecting screenshots accessible via www-i7.informatik.rwth-aachen.de/∼wallmeier.

The following table summarizes the computation results on the lift controller example with small floor numbers $e = 3, 4, 5$. The computation was done on an Intel P3-M 1GHz with 256MB memory.

Table 1. Results of the case study

floors	vars	RR-pairs	states	BDD-time	Büchi-states	game-time	W_0	W_1
3	15	3	25	18.34s	1,200	24.27s	24	1
4	18	6	673	27.73s	516,864	40.39s	672	1
5	23	9	12,913	40.86s	119,006,208	699.34s	0	12,913

The table shows first the numbers of floors, of introduced Boolean variables, of RR-pairs, of states and the time to compute the BDD representation (using the package BuDDy of Jørn Lind-Nielsen [2]). The dramatic (exponential) increase of states for the corresponding Büchi game is clear from the next column. The entry "game-time" refers to the reduction to Büchi games and their solution. The last two columns show the sizes of the winning regions. For $e = 3, 4$ the controller wins (so here the specification is satisfiable) except from the special sink state. Already for 5 floors, however, controller is beaten (its winning region is empty). An analysis of environment's winning strategy exhibits the problem for controller: The environment forces one lift to the second floor (post office) without any other request. This lift is blocked in the next turn of the controller. The environment can now requests two floors, one of them an express floor, the other one on the way to it. Now the controller can not satisfy requests 2. and 5. simultaneously.

6 Conclusion

We have described the algorithmic solution of a class of infinite games and applied this for synthesis of finite state-controllers. The implementation offers an input language based on parameterized Boolean logic, and uses logical specifications which are given by conjunctions of RR-conditions. This is in contrast to the standard games of theoretical interest, where the winning conditions are usually given as disjunctions (rather than conjunctions). To our knowledge the implementation is the first program which allows to study the controller synthesis problem using the algorithmic theory of infinite games.

From our experiments (as in the case of the lift controller) it is clear that the number of RR-conditions is the main bottleneck. The same effect is to be expected if other conjunctive winning conditions (like Streett conditions) are used. How to dissolve the blow-up involved in the reduction of RR-games to Büchi games? The obvious approach would be to treat the RR-conditions separately. In the present example, it happens that the negative result for $e = 5$ could be obtained already by considering a subset of the request-response-conditions (namely, by forgetting the requests from inside the lifts and only taking into account the "outside requests"). This decomposition cannot, however, work in general: Simple examples show that the conjunction of request-response-conditions c_1, c_2 cannot be handled by solving the games for c_1, c_2 individually and proceeding to the intersection of the two computed winning regions for player 0. Even if the winning region is obtained as such an intersection, the winning strategy cannot, in general, be computed in this way but has to be generated globally. Ongoing work tries to clarify under which assumptions such conjunctions can be treated incrementally (one by one), and how to support the user to keep the number of these conditions small.

Acknowledgment

We thank a referee for his remark that a lower bound for RR-games should be given (now provided in Theorem 2).

References

[1] E. Clarke, O. Grumberg, and D. Peled. *Model Checking.* The MIT Press, Cambridge, Massachusetts, 1999.

[2] J. Lind-Nielsen. Buddy - http://www.it-c.dk/research/buddy/.

[3] H. Melcher. *Hierarchische kompositionale Synthese von Steuerungen für reaktive Systeme*. PhD thesis, Uni Karlsruhe, 2001.

[4] D. Perrin and J.-E. Pin. *Infinite Words (to appear)*. Elsevier, http://www.liafa.jussieu.fr/~jep/Resumes/InfiniteWords.html, 2003.

[5] D. Schmitz and J. Vöge. Implementation of a strategy improvement algorithm for parity games. In *Proceedings of the 5th International Conference on the Implementation and Application of Automata, CIAA 2000*, pages 45–51, 2000.

[6] W. Thomas. On the synthesis of strategies in infinite games. *Lecture Notes in Computer Science*, 900:1–13, 1995.

[7] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume III, pages 389–455. Springer, New York, 1997.

[8] W. Thomas. Infinite games and verification. In *Proceedings of the International Conference on Computer Aided Verification CAV'02*, volume 2404 of *Lecture Notes in Computer Science*, pages 58–64. Springer, 2002.

[9] J. Vöge and M. Jurdziński. A discrete strategy improvement algorithm for solving parity games. In *Proceedings of the 12th International Conference on Computer Aided Verification, CAV 2000*, volume 1855 of *Lecture Notes in Computer Science*, pages 202–215. Springer, 2000.