ELSEVIER

Theoretical Computer Science 194 (1998) 183–205

Theoretical Computer Science

Resolution-based approach to compatibility analysis of interacting automata

Anatoli N. Chebotarev, Marina K. Morokhovets*

Glushkov Institute of Cybernetics, Ukrainian Academy of Sciences, Glushkov prosp., 40, Kiev 252187, Ukraine

Received December 1994; revised May 1996
Communicated by A.A. Letichevsky

Abstract

The problem of compatibility analysis of two interacting automata arises in the development of algorithms for reactive systems if partial nondeterministic automata serve as models for both the reactive system and its environment. Intuitively, two interacting automata are compatible if a signal from one automaton always induces a defined transition in the other. We present the method for solving this problem in the automated design of reactive system algorithms specified by formulas of a first-order monadic logic. Compatibility analysis is automatically performed both for initial specifications of the algorithm and its environment and for any specification of the same kind obtained as a result of changes made by the designer. So, there is a need in the development of methods for solving this problem at the level of logical specification. The methods based on the resolution principle proved to be very helpful for this purpose.

1. Introduction

The problem considered in this paper relates to the development of reactive systems [12, 15], i.e. the systems whose operation is continuous interaction with the environment. Typical examples of reactive systems are telecommunication networks, patient monitoring systems, process control systems, and others. For a wide class of such systems, nondeterministic finite automata are used as mathematical models of both the system under development and its environment (see e.g. [3, 13, 11, 14]). Here we restrict our attention to synchronous sequential systems which are modeled by finite deterministic automata. In contrast to automaton models in the theory of concurrent systems, such as Buchi or Muller automata, we do not think of an automaton as an acceptor but rather as a transformer that transforms infinite input sequences into

* Corresponding author. E-mail: mmk@d105.icyb.kiev.ua.

PII S0304-3975(97)00025-X

infinite output sequences. An essential property of such automata is their "cyclicity", i.e., nonexistence of the state in which the automaton's operation terminates. As a result, they can operate for an infinitely long time.

A semantics of a reactive system specification is described in the form of two interacting nondeterministic automata. One of them specifies the behaviour of the system under development and the other specifies the behaviour of the system's environment, or more precisely, available information about the behaviour of this environment. These interacting automata form a structure in which the inputs and outputs of one automaton are connected, respectively, to the outputs and inputs of the other automaton. The partiality of the automata (one or both) has an essential impact on the possible behaviour of this structure. If the interacting automata are partial, the problem arises to ensure correct joint operation of the two automata, i.e. to eliminate the cases where the transition in one automaton caused by the signal received from the other automaton is undefined. The corresponding problem, which arises in the development of reactive systems, can be stated as follows: using information about the environment, design the system so as to ensure correct joint operation of the system and the environment.

The solution to this problem is significantly influenced by nondeterminism of the automata. This nondeterminism comes from lack of information about the system's behaviour. In other words, it is viewed as incompleteness of the specification which results in that more than one deterministic automaton satisfies, in a sense, the specification. So, a nondeterministic automaton is treated as a set of deterministic automata. It should be noted that we interpret nondeterminism of the system under design and nondeterminism of its environment in different ways. We regard the nondeterminism of the system as the possibility to make a choice among different deterministic implementations during the design process, i.e. any deterministic automaton which satisfies the specification is a solution to the design problem. In contrast, nondeterminism of the environment demands to consider all its possible behaviours resulting from this nondeterminism. If the specification of the environment is empty (there is no environment specification) which corresponds to a fully nondeterministic model, then it is natural to expect that the environment can behave in an arbitrary way. Such a difference in interpreting nondeterminism of an automaton and its environment brings asymmetry in the consideration of their interaction.

In order to refine and formalize the concept of correct interaction of automata, the notion of compatibility of automata was introduced in [5]. Compatibility is defined as a property of the cyclic composition of automata that specifies the appropriate mode of their interaction. As we indicated previously, the specification of an automaton usually defines a class of automata and the design problem is to construct a particular automaton from this class. The information about the environment restricts the choice of the representative automaton from this class by the requirement for this automaton to be compatible with the environment. So, the solution to the compatibility problem not only must answer the question whether the automata are compatible or not but also distinguish in the set of all automata satisfying the specification a subset of automata compatible with the environment.

It is obvious that the compatibility problem should be solved at the highest level of the design process, i.e. at the level of specifications. The presented solution to this problem is intended for the implementation within the methodology of the design of reactive systems from their logical specifications, when the first-order monadic logic is used as a specification language [7]. Given the specification of this kind, the important problem is to check the system specification for consistency. However, if the specification is partial, then this is not sufficient, we have also to acsertain that it is compatible with the environment specification. The efficient algorithm for verifying whether an automaton specification expressed as a formula of the first-order monadic logic is consistent was obtained in [7]. This algorithm is based on the resolution inference search procedure. In this paper, the results of [7] and the technique for compatibility analysis developed in [5] are used to construct the procedure that solves the compatibility problem using the data representation described in [7].

The main goal of [7] and the present paper was to obtain a technique that avoids the consideration of all the states of the automata composition. The method presented in this paper makes it possible to consider states of only one of interacting automata, namely, the automaton describing the behaviour of the system under design. Nondeterminism of the environment model essentially complicates the compatibility analysis. To simplify the problem we consider it for the case where interacting systems are modeled by automata with finite memory [10] (which constitute a subclass of finite automata). This restriction of the class of automaton models does not particularly limit the applicability of the results, because most implementations of discrete systems described by automata are, in fact, in the class of finite-memory automata. Further, we state and solve the compatibility problem for noninitial automata though an initial automaton is a more appropriate model. In the methodology of the reactive system design, an initial automaton is represented as a pair $\langle A, \varphi \rangle$, where A is a noninitial automaton and φ is a condition distinguishing initial states. The identification of the initial state is made at the subsequent stages of the design which enables us to simplify the procedures for solving such design problems as analysis of automata specifications for consistency and compatibility. Finally, it should be noted that though the automaton model is obviously insufficient to describe real systems that perform complicated transformations of structured data, nevertheless the technique proposed in this paper can be successfully used in the design of the control part of such systems, which is responsible for the interaction with the environment.

The rest of the paper is organized as follows. In Section 2, the basic notions and results concerning the compatibility analysis of automata are presented. Although the paper deals with the development of sequential systems, the problems which arise when considering their interaction with the environment are much in common with those for concurrent systems. So, in Section 2.3, the notion of compatibility is compared with the related notions in the theory of concurrent systems. In Section 3, we outline the logical language for automata specifications and describe the correspondence between formulas of this language and specified automata. When solving the compatibility problem, we consider some set of determinizations of a nondeterministic automaton. So, in Section 4,

the technique for constructing the appropriate determinizations of an automaton specified as a set of clauses is described. In the last section, an approach to the solution of the compatibility problem based on resolution is presented and its soundness and completeness is proved.

2. Compatibility of automata: The basic notions and preliminary results

In this section, we briefly review the basic notions and some results concerning the automata compatibility checking problem. For more details the reader is referred to [5].

2.1. Defining compatibility

Definition 1. *A finite* $(X-Y)$*-automaton* A is a 5-tuple $A=\langle X, Y, S_A, \chi_A, \mu_A\rangle$, where X, Y are, respectively, input and output finite alphabets, S_A is a finite set of states; $\chi_A : S_A \times X \to 2^{S_A}$ and $\mu_A : S_A \to Y$ are, respectively, transition and output functions. The automaton A is called *partial* if there exists such $x \in X$ and $s \in S_A$ that $\chi_A(s,x)=\emptyset$. The automaton A is called *cyclic* if for each $s \in S_A$ there exist $s_1, s_2 \in S_A$ and $x_1, x_2 \in X$ such that $s_1 \in \chi_A(s,x_1)$ and $s \in \chi_A(s_2,x_2)$. The automaton A is called *deterministic* if for all $x \in X$ and $s \in S_A$ $|\chi_A(s,x)| \leqslant 1$; otherwise A is called *nondeterministic*. If $|X|=1$, then the automaton A is called *an autonomous* Y-automaton.

A finite sequence over the alphabet X will be referred to as X*-word*. If X and Y are, respectively, the input and output alphabets of an automaton then an $(X \times Y)$-word will also be referred to as an *input–output word*.

Definition 2. An $(X \times Y)$-word $l=(x_0,y_0)(x_1,y_1)\ldots(x_k,y_k)$ is *admissible in the state* s_0 of the $(X-Y)$-automaton A if there exists a sequence of states $s_0, s_1, \ldots, s_{k+1}$ such that for all $i=0,1,\ldots,k$, $s_{i+1} \in \chi_A(s_i,x_i)$ and $\mu_A(s_i)=y_i$. Any state word that satisfies these conditions is called *corresponding* to the input–output word l. An $(X \times Y)$-word l is *admissible* for the $(X-Y)$-automaton A if it is admissible in some state of this automaton.

The definition of admissibility of $(X \times Y)$-words is extended to superwords (infinite words): a superword $(x_0,y_0)(x_1,y_1)\ldots$ is admissible iff its every finite prefix is admissible. The set of all superwords admissible in the state s is denoted by $L(s)$.

Definition 3. States s_1 and s_2 of the same automaton or different automata are equivalent if $L(s_1)=L(s_2)$.

Definition 4. Automata A_1 and A_2 are equivalent if for every state of one of them there exists an equivalent state of the other and vice versa.

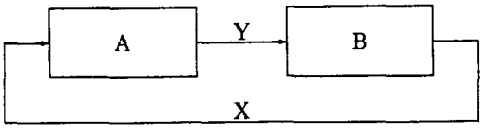

Fig. 1.

To formalize the concept of compatibility of the $(X-Y)$-automaton A with the $(Y-X)$-automaton B, the notion of their *cyclic composition* is introduced (cf. [5]). This composition is an autonomous, nondeterministic automaton C whose state set is $S_A \times S_B$ and the output alphabet is $X \times Y$. The transition function χ_C and the output function μ_C of this automaton are defined as follows. For all $s' \in S_A$, $s'' \in S_B$

$$\chi_C(s',s'') = \{(s_1',s_1'') \mid s_1' \in \chi_A(s',\mu_B(s'')),\ s_1'' \in \chi_B(s'',\mu_A(s'))\},$$

$$\mu_C(s',s'') = (\mu_A(s'),\mu_B(s'')).$$

Intuitively, the automaton C corresponds to the composition shown schematically in Fig. 1.

Definition 5. Two cyclic, partial, deterministic $(X-Y)$- and $(Y-X)$-automata are called *compatible* if their cyclic composition has a cyclic subautomaton.

Since the operation of the composite automaton C may cease only when one of the component automata enters such a state s for which the transition caused by the current input symbol x is undefined ($\chi(s,x)=\emptyset$), Definition 5 means that there exists such a pair of component automata's states that starting from the corresponding state the automaton C never encounters this situation.

In order to define the corresponding notion for nondeterministic automata, the concept of a determinization of a nondeterministic automaton is introduced. This concept is based on the formal notions of a transition and its determinization.

Definition 6. A transition from a state s caused by an input symbol x in a nondeterministic $(X-Y)$-automaton A is a triple $\langle s,x,\chi_A(s,x)\rangle$, where $\chi_A(s,x) \neq \emptyset$. A transition is called *nondeterministic* if $|\chi_A(s,x)| > 1$ and *deterministic* if $|\chi_A(s,x)| = 1$.

The automaton A can be viewed as a set of transitions corresponding to all pairs $\langle s,x\rangle$ $(s \in S_A, x \in X)$ such that $\chi_A(s,x) \neq \emptyset$.

A determinization of the transition $\langle s,x,\chi_A(s,x)\rangle$ is the triple $\langle s,x,s'\rangle$, where $s' \in \chi_A(s,x)$.

Definition 7. *A basic determinization* of a nondeterministic automaton A is an automaton obtained by replacing each transition in A with some determinization of this transition.

So, any basic determinization of the automaton A has the same state set as A. Every basic determinization of a cyclic automaton A has a nonempty cyclic subautomaton. The largest such subautomaton is called a *basic cyclic determinization* of the automaton A.

Definition 8. A basic determinization of any automaton equivalent to the automaton A is called *a determinization* of the automaton A.

We similarly define the notion of cyclic determinization of a cyclic nondeterministic automaton. From Definition 8 it follows that the set of all determinizations (cyclic determinizations) of the automaton A is countably infinite, because so is the set of all automata equivalent to A.

Definition 9. A partial, nondeterministic, cyclic $(X-Y)$-automaton A is compatible with a partial, nondeterministic, cyclic $(Y-X)$-automaton B iff there exists a cyclic determinization of the automaton A compatible with each cyclic determinization of the automaton B.

The significance of this definition for design problems is that if A is a partial automaton under design and B is the automaton characterizing its environment, then A must be compatible with B.

2.2. Compatibility theorems

The direct use of Definitions 5 and 9 to construct the compatibility analysis algorithm is not only inconvenient but also impossible. Even if the automaton B that represents the environment is deterministic, the compatibility analysis requires considering all the states of the cyclic composition of two automata. If the automaton B is nondeterministic, then Definition 9 is nonconstructive at all, since it uses the infinite set of all determinizations of the automaton B. The results obtained in [5] enables us, first, to restrict the number of the determinizations to be considered to finite and not very large amount; second, to consider only the states of the automaton A when checking the compatibility of the automaton A with the automaton B. This is achieved at the expense of restricting the class of the automata involved. In [5], the problem whether a partial, nondeterministic, cyclic automaton A is compatible with a completely defined, nondeterministic, cyclic automaton B is solved for the case where A and B are automata with finite memory [10].

Definition 10. An automaton A is called an automaton *with finite memory* if there exists a natural δ such that for any admissible input–output word $(x_0, y_0)(x_1, y_1)\dots(x_{k-1}, y_{k-1})$ of length $k \geqslant \delta$ and any two corresponding state words $s'_0 s'_1 \dots s'_k$ and $s''_0 s''_1 \dots s''_k$ either the states s'_k and s''_k are equivalent, or $\mu_A(s'_k) \neq \mu_A(s''_k)$. The minimal such δ is called the *depth of the memory*.

Note that in this definition δ may be equal to 0, which corresponds to the empty input–output word. For this word any state forms the corresponding state word. This implies that in the case of $\delta = 0$, for any two inequivalent states s_1, s_2 of the automaton A, $\mu_A(s_1) \neq \mu_A(s_2)$.

In order to describe the results obtained in [5], we need some additional notions. A special form of representing automata with finite memory is considered. For each $(X - Y)$-automaton A with finite memory of depth δ and each natural $k \geqslant \delta$, there exists an $(X - Y)$-automaton A^* equivalent to A such that for any admissible input–output word of length k and any two corresponding state words $s'_0, \ldots s'_k$ and $s''_0, \ldots s''_k$, either $s'_k = s''_k$ or $\mu_{A^*}(s'_k) \neq \mu_{A^*}(s''_k)$, and for different input–output words l_1, l_2 of length k, any two state words corresponding, respectively, to l_1 and l_2 end with different states. The automaton A^* will be called *k-normal form* of the automaton A. In the k-normal form of a cyclic automaton A, any state s is uniquely determined by the pair $(l, \mu_A(s))$, where l is an input–output word of length k. In its turn, a state s is uniquely associated with an input–output word l_s of length k.

Definition 11. An $(Y \times X)$-word $(y'_0, x'_0) \ldots (y'_k, x'_k)$ is called *a reflection* of an $(X \times Y)$-word $(x_0, y_0) \ldots (x_k, y_k)$ if $x_i = x'_i$, $y_i = y'_i$ for all $i = 0, \ldots, k$.

Let δ_A be the depth of the memory of the automaton A, δ_B the same for B, $\delta = \max(\delta_A, \delta_B)$, and let A^* and B^* be δ-normal forms of the automata A and B. The state s_1 of the automaton A^* and the state s_2 of the automaton B^* are called *the reflections* of each other if the corresponding input–output words l_{s_1} and l_{s_2} are the reflections of each other.

We are now ready to state the main results underlying the technique for compatibility analysis of automata.

Theorem 12. *Let B be a deterministic automaton, A^* and B^* be δ-normal forms of the automata A and B, and C^* be the subautomaton of A^* formed by all the states of A^* which are the reflections of the states of B^*. The automaton A is compatible with the automaton B iff C^* has a cyclic subautomaton.*

Theorem 13. *A partial, nondeterministic, cyclic $(X - Y)$-automaton A with finite memory is compatible with a nondeterministic, cyclic $(Y - X)$-automaton B with finite memory iff it is compatible with each basic cyclic determinization of the δ-normal form of the automaton B.*

2.3. Related work

The notion similar to the notion of compatibility of an automaton with its environment in the case of no information about the environment is available, is used by Abadi and Lamport [2] for a more general model of a system. This is a notion of *receptiveness*.

To clarify the relation between these two notions, observe that an input–output superword $(x_0, y_0)(x_1, y_1)\ldots$ may be regarded as a *behaviour* (in the sense of [2]) if inputs and outputs of the automaton is viewed as externally visible state components [1] and each pair of successive symbols $(x_i, y_i)(x_{i+1}, y_{i+1})$ of the input–output superword is associated with two steps of the corresponding behaviour

$$(x_i, y_i) \to^{x_{i+1}} (x_{i+1}, y_i) \to^{y_{i+1}} (x_{i+1}, y_{i+1}),$$

where x_{i+1}, y_{i+1} over the arrows stand for environment and system agents, respectively. Thus, the set of all admissible for $(X - Y)$-automaton A input–output superwords corresponds to the property associated with this automaton. More precisely, the property is the closure of the corresponding set of behaviours under stuttering equivalence [2]. The following statement shows the relation between the two notions. An automaton A is completely defined (i.e. compatible with any environment) iff the associated property is receptive.

Moreover, if in defining the notion of realizability in [2] the environment's behaviour would be thought of as restricted by its specification, then the *realizable part* of a set P_A of behaviours associated with the automaton A would correspond to the result of checking the compatibility of the automaton A with the environment.

The notion analogous to compatibility of an automaton with its environment is considered by Cleaveland and Steffen [9] for processes in the setting of Milner's CCS, where divergency is viewed as partiality. In their paper such a notion is called adequacy, and the key problem is to answer the question when a (finite-state) partial process specification is adequate, or "complete enough" for the context in which it is to be used. They use the mu-calculus formula generated for the specification of a process to characterize the context for which this specification is adequate. The methodology described in [9] as well as in our case avoids the construction of the state space of the composite process.

The notion of compatibility of a nondeterministic automaton with its environment is asymmetric because of the difference in interpreting nondeterminism of the automaton and the environment. In the case where nondeterminism of the interacting automata is interpreted uniformly, it is natural to consider a symmetric notion of compatibility. This notion can be introduced, for example, in the manner it is made in [5], where the notion of *mutual compatibility* of automata has been introduced. This notion can be generalized for the case of a network of automata. The corresponding notion for the network of communicating processes is considered by Chen et al. [8], where it is called *an absence of computation interference*. A similar notion, called mutual adequacy, has been also introduced in [9].

3. Formulas and automata

A specification S of an automaton is a description in a formal language of the class $K(S)$ of equivalent nondeterministic automata. If S is viewed as a specification

of a deterministic automaton, then the class of all automata satisfying S (specified by S) is a class of all determinizations of any automaton in $K(S)$. If S is viewed as a specification of a nondeterministic automaton then the class of all automata satisfying S coincides with $K(S)$.

As the specification language, we use a subset of the first-order language with monadic predicates interpreted over the set of integers which is regarded as a discrete time domain (cf. [7]).

Let Φ be a class of formulas constructed by means of logical connectives from atoms of the form $p(t+k)$, where p is a unary predicate symbol, t is a variable, and k is an integer constant, called *the rank of the atom*. The *rank of a formula* $F(t) \in \Phi$ is the difference between the maximal and minimal ranks of atoms occurring in $F(t)$.

The specification S of a cyclic automaton with finite memory is of the form $\forall t F(t)$ [7], where $F(t) \in \Phi$. Consider one way to construct a series of representative automata from $K(S)$. The predicate symbols occurring in $F(t)$ correspond to the input and output channels of automata from $K(S)$, so they are partitioned into input and output symbols. Let $U = \{u_1, \ldots, u_m\}$ be the set of input predicate symbols and $W = \{w_1, \ldots, w_n\}$ the set of output predicate symbols occurring in $F(t)$. We define the input and output alphabets of the $(X-Y)$-automata specified by the formula $\forall t F(t)$ as the sets of all binary vectors of length m and n, respectively.

We next define the notion of state for the formula $F(t)$. Since $F(t)$ is interpreted over the set of integers the equivalence $\forall t F(t) \Leftrightarrow \forall t F(t+k)$, where $F(t+k)$ denotes the formula obtained from $F(t)$ by adding k to the ranks of all its atoms, holds for any integer k. So, we may assume that the maximal rank of atoms occurring in $F(t)$ is equal to 0. A formula satisfying this condition will be called *normal*, and the replacement of a formula with the equivalent normal formula will be called *normalization*. A normal formula for which all atoms of maximal rank are output atoms will be referred to as a *proper* formula. It can be easily shown that for every formula $\forall t F(t)$ there exists an equivalent formula $\forall t F'(t)$, where $F'(t)$ is a proper formula.

Let $F(t)$ in the specification S be a proper formula and r_F be the rank of this formula. For each $r \geqslant r_F$, a sequence $(y_0, x_0, y_1, x_1, \ldots, x_{r-1}, y_r)$, where $y_i \in Y$ $(i = 0, \ldots, r)$ and $x_j \in X$ $(j = 0, \ldots, r-1)$, is called *a state of rank* r and a set $\Sigma(r, F(t))$ of all such sequences will be referred to as *a state space of rank* r for the formula $F(t)$. We define a function N from $\Sigma(r, F(t))$ to $2^{\Sigma(r, F(t))}$. Given $s = (y_0, x_0, \ldots, x_{r-1}, y_r)$, $N(s)$ is defined as the set of 2^{m+n} states of the form $(y_1, x_1, \ldots, y_r, x, y)$, where $x \in X$, $y \in Y$. States in $N(s)$ will be referred to as successors of s.

If components of the vectors y_i $(i = 0, \ldots, r)$ and x_j $(j = 0, \ldots, r-1)$ in the state $s = (y_0, x_0, \ldots, x_{r-1}, y_r)$ are viewed as the truth values of the corresponding (w.r.t. some orderings on W and U) atoms of rank $i - r$ and $j - r$, respectively, then we can evaluate the formula $F(t)$ on the state s in the standard way.

Example 14. Let $F(t)$ be of the form $(w_1(t-2) \& \neg u(t-1) \vee \neg w_2(t-2) \& w_1(t-1)) \rightarrow w_1(t)$ and $U = \{u\}$, $W = \{w_1, w_2\}$. The rank of this formula equals 2. Evaluate $F(t)$ on the state $(y_0, x_0, y_1, x_1, y_2) = (\langle 01 \rangle, \langle 1 \rangle, \langle 11 \rangle, \langle 0 \rangle, \langle 10 \rangle)$, assuming that the

output predicate symbols are ordered according to their order in the specification of W. Output atoms of rank 0 take on values from y_2, input and output atoms of rank -1 from x_1 and y_1, respectively, and so on. Thus, $w_1(t)=1$, $u(t-1)=0$, $w_1(t-1)=1$, $w_1(t-2)=0$, $w_2(t-2)=1$, and hence the value of $F(t)$ on the state (y_0,x_0,y_1,x_1,y_2) equals **1**.

Every state space $\Sigma(r,F(t))$ is associated with a representation $A(r,F(t))$ of a nondeterministic automaton from the equivalence class corresponding to the formula $\forall tF(t)$. $A(r,F(t))$ is a maximal cyclic subautomaton of an automaton $A'(r,F(t))$ defined as follows. The state set of $A'(r,F(t))$ is the set $S_{A'}$ of all that states in $\Sigma(r,F(t))$ on which $F(t)$ is true. The transition and output functions of $A'(r,F(t))$ are defined as follows: $\chi_{A'}(s,x)$ is the set of all states $(y_0,x_0,\ldots,x_{r-1},y_r)$ in $N(s)\cap S_{A'}$ for which $x_{r-1}=x, \mu_{A'}(s=(y_0,x_0,\ldots,x_{r-1},y_r))=y_r$. Since a state $s=(y_0,x_0,\ldots,x_{r-1},y_r)$ of the automaton $A'(r,F(t))$ is uniquely determined by the pair consisting of the input–output word $(x_0,y_0)\ldots\ldots(x_{r-1},y_{r-1})$ of length r and $y_r=\mu_{A'}(s)$, this representation is an r-normal form.

If the proper form of $F(t)$ does not contain output atoms of minimal rank, then we may consider states of the form $(x_0,y_1,x_1,\ldots,x_{r-1},y_r)$. Such a state will be called a state of rank $r-0.5$, and the automaton A' (A) associated with the state space $\Sigma(r-0.5,F(t))$ will be referred to as $(r-0.5)$-normal form.

Example 15. Let $F(t)$ in the automaton specification S be of the form $(w(t-1)\rightarrow u(t-1)\,\&\,\neg w(t))\,\&\,(\neg u(t-2)\,\&\,\neg u(t-1)\rightarrow\neg w(t))\,\&\,(u(t-2)\,\&\,\neg u(t-1)\rightarrow w(t))$, where u and w are input and output predicate symbols, respectively. We represent the state space $\Sigma(1.5,F(t))$ in the form of a square table (a K-map) each entry of which corresponds to a state of rank 1.5 viewed as a value of the vector of boolean variables $\langle u(t-2),w(t-1),u(t-1),w(t)\rangle$. The K-map specifying the state space for this example with the graph of the automaton $A(1.5,F(t))$ superimposed is shown in Fig. 2. The dots on the K-map denote the states on which $F(t)$ is true. The depicted automaton

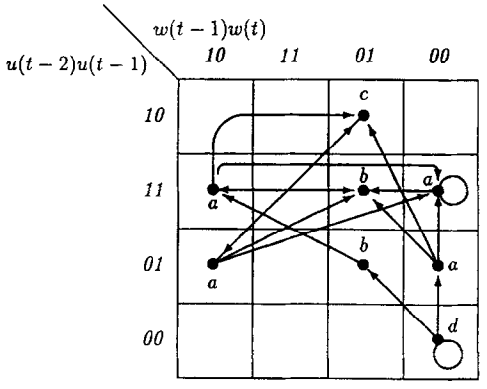

Fig. 2.

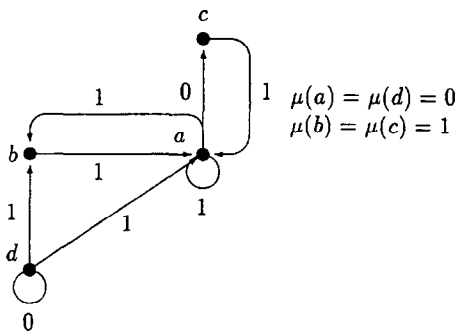

Fig. 3.

is a 1.5-normal form of the automata in $K(S)$. The corresponding 1-normal form of this automaton is shown in Fig. 3. In Fig. 3, the arcs (the transitions) are labelled with symbols of the input alphabet $X = \{0, 1\}$, and states are designated by the letters a, b, c, d. These letters mark the states of the automaton in Fig. 2. Identical letters mark the states equivalent to the state designated by the same letter in Fig. 3.

The formula $F(t)$ in the automaton specification is assumed to be represented in CNF (conjunctive normal form), which we regard as a set of clauses, i.e. disjunctions of literals, where a literal is an atom or its negation.

The rank of a literal is defined as the rank of the corresponding atom. The rank of a clause set $\{c_1, \ldots, c_n\}$ is the rank of the formula $c_1 \,\&\, \ldots \,\&\, c_n$.

In what follows, we consider an $(X - Y)$-automaton A and $(Y - X)$-automaton B as the models of the reactive system being designed and its environment, respectively. The automaton A is a partial, deterministic, cyclic automaton with finite memory satisfying the initial specification $\forall t F_A(t)$ represented as a set of clauses C_A. The automaton B is essentially nondeterministic. Its nondeterminism results from incompleteness of knowledge about the environment. This automaton is specified by the formula $\forall t F_B(t)$ represented as a set of clauses C_B. The formulas $F_A(t)$ and $F_B(t)$ are assumed to have the same signature of predicate symbols $\Omega = U \cup W$, where U and W are the sets of input and output (w.r.t. $F_A(t)$) predicate symbols, respectively.

Throughout the sequel, a literal corresponding to a predicate symbol in $U(W)$ will be referred to as an $U(W)$-literal.

We shall consider two classes of clause sets: sets of normal clauses and sets of proper clauses. A clause c is called normal if the maximal rank of its literals is 0. A clause c is called proper if it is either normal and contains no input literals of rank 0 or the maximal rank of its literals equals -1 and it contains input literals of this rank. Clearly, a set of normal clauses represents a normal formula, and a set of proper clauses containing at least one normal clause represents a proper formula. A clause c' obtained from a clause c by adding an integer k to the ranks of all its literals is called a k-shift of the clause c. Since any clause c represents a formula of the form $\forall t f(t)$,

we can equivalently replace c with its k-shift for an arbitrary integer k. Hence any clause set C can be equivalently transformed to a set of normal clauses as well as to a set of proper clauses. Sometimes, for a set of proper clauses C we shall consider the corresponding set of normal clauses that is obtained by normalizing every clause in C.

4. Determinizations

In this section, we investigate the criterion of determinism and methods for constructing determinizations of an $(Y-X)$-automaton specified by a set of proper clauses. If the formula $F_B(t)$ in the specification of the automaton B is represented by the set of clauses C_B of rank r_B, then the corresponding nondeterministic automata in the state space $\Sigma(r, F_B(t))$, $r \geqslant r_B$, will be denoted by $B'(r, C_B)$ and $B(r, C_B)$.

We are interested in the set of all basic cyclic determinizations of the r-normal form of the automaton $B(r, C_B)$, where $r \geqslant \max(r_A, r_B)$. Every basic cyclic determinization of $B(r, C_B)$ is uniquely determined by the corresponding determinization of this automaton. In its turn, every determinization of $B(r, C_B)$ is induced by some determinization of $B'(r, C_B)$. However, in general, not every determinization of the automaton $B'(r, C_B)$ induce the determinization of $B(r, C_B)$. Below, we shall state the condition under which every determinization of $B'(r, C_B)$ uniquely determines a determinization of $B(r, C_B)$. With this in mind consider how to construct the set of basic determinizations of the automaton $B'(r, C_B)$. Since the automata $B'(r, C_B)$ and $B(r, C_B)$ are specified by a set of clauses, we introduce the notions of determinism and nondeterminism for clause sets.

Let $s = (x_0, y_0, \ldots, y_{r-1}, x_r)$, then $N(s) = \{(x_1, y_1, \ldots, x_r, y, x) \mid y \in Y,\ x \in X\}$. Denote by $N(s, y')$ the subset of $N(s)$ obtained by fixing $y = y'$. This subset will be called *a transition domain* for $s \in \Sigma(r, F_B(t))$ and $y' \in Y$.

Definition 16. Let $N_B(s, y')$ be the set of all states in $N(s, y')$ on which the formula $F_B(t)$ specified by the clause set C_B is true. The set of clauses C_B is *deterministic (nondeterministic) on the domain* $N(s, y')$ if $|N_B(s, y')| = 1$ ($|N_B(s, y')| > 1$) and is *undefined on* $N(s, y')$ if $N_B(s, y') = \emptyset$.

Determinism (nondeterminism) of C_B on the domain $N(s, y') \subseteq \Sigma(r, F_B(t))$ corresponds to determinism (nondeterminism) of the transition $\langle s, y', \chi_{B'}(s, y')\rangle$ in the automaton $B'(r, C_B)$.

Definition 17. A set of clauses C_B is deterministic if, on every domain $N(s, y)$ ($s \in \Sigma(r, F_B(t))$, $y \in Y$), C_B is either deterministic or undefined.

Definition 18. *A determinization of a set of clauses* C_B of rank r_B is a deterministic set of clauses C_B^* of rank $r^* \geqslant r_B$ such that $B'(r^*, C_B^*)$ is a determinization of $B'(r, C_B)$.

Replacement of a nondeterministic transition $\langle s, y, \chi_{B'}(s, y)\rangle$ in the automaton $B'(r, C_B)$ $(r \geqslant r_B)$ with its determinization corresponds to replacement C_B with the clause set C'_B such that the formula $F'_B(t)$ specified by C'_B is true on a single state in $N(s, y)$ and on the states outside this domain its values are equal to the values of $F_B(t)$. The set C'_B can be obtained by adding some clauses to C_B. Since a basic determinization of the automaton $B'(r, C_B)$ is obtained by the replacement of every nondeterministic transition with its determinization and $F^*_B(t)$ specified by a clause set $C^*_B \supseteq C_B$ is only true on that states in $\Sigma(r, F_B(t))$ on which $F_B(t)$ is true, we may state the sufficient condition for C^*_B to be a determinization of C_B as follows:

(1) C^*_B is deterministic;

(2) on every domain $N(s, y) \subseteq \Sigma(r, F_B(t))$ where C^*_B is undefined C_B is also undefined.

In order to describe the method for constructing determinizations of a nondeterministic set of proper clauses C_B, the following results are needed.

A formula $F_B(t)$ specified by a set of clauses C_B will be viewed as a propositional formula whose variables are atoms of $F_B(t)$. Let the rank of $F_B(t)$ be r_B and $r \geqslant r_B$. Consider the expansion of the formula $\neg F_B(t)$ of the form $\varphi_1 \,\&\, \alpha_1 \vee \varphi_2 \,\&\, \alpha_2 \vee \cdots \vee \varphi_N \,\&\, \alpha_N$, where φ_i are all minterms[1] for the variables corresponding to all atoms of ranks $-1, \ldots, -r$ for the signature Ω, and α_i are formulas represented in disjunctive normal form, whose variables correspond to (output) atoms of rank 0. It is worth noting that every minterm φ_i $(i = 1, \ldots, N)$ determines a transition domain $N(s, y)$ for some $s \in \Sigma(r, F_B(t))$ and $y \in Y$ in the sense that φ_i is only true on the states of this domain, and $\varphi_i \,\&\, \alpha_i$ determines $\neg N_B(s, y) \,\&\, N(s, y)$.

Definition 19. The above expansion of the formula $\neg F_B(t)$ *is consistent* with C_B if

(a) for every elementary conjunction d in α_i $(i = 1, \ldots, N)$, there is a clause in C_B which contains the negations of all literals of d and subsumes[2] the clause equivalent to $\neg(\varphi_i \,\&\, d)$;

(b) for every clause $c \in C_B$, there exists φ_i and an elementary conjunction d in α_i whose literals are the negations of all literals of rank 0 occurring in c such that c subsumes the clause equivalent to $\neg(\varphi_i \,\&\, d)$.

A normal clause which contains exactly one literal of rank 0 will be called *singular*. Depending on whether the literal of rank 0 is W- or U-literal a singular clause will be referred to as a *W- or U-clause*, respectively.

In what follows, we will assume that all normal clauses in C_B are singular. The method for constructing determinizations of a clause set with nonsingular clauses is given in [6], but in this case compatibility analysis of automata is more laborious.

Proposition 20. *In the expansion of the formula $\neg F_B(t)$ consistent with C_B every α_i is a disjunction of U-literals of rank* 0.

[1] A minterm for a set of propositional variables $\{u_1, \ldots, u_n\}$ is an elementary conjunction containing for every $i = 1, \ldots, n$ either u_i or $\neg u_i$.

[2] A clause c_1 subsumes a clause c_2 if c_1 is a subset of c_2.

Proposition 21. *Let C_B be a deterministic set of clauses. In the expansion of $\neg F_B(t)$ consistent with C_B, every α_i either contains a pair of complementary literals or consists of m U-literals: one literal for every predicate symbol in U. Conversely, if the expansion of $\neg F_B(t)$ possesses the above properties, then C_B is deterministic.*

This proposition follows from Definitions 16, 17, 19, and the fact that every φ_i in the expansion determines a transition domain for some s in the state space $\Sigma(r, F_B(t))$ and $y \in Y$. Intuitively, if α_i contains a pair of complementary literals, then it corresponds to a transition domain on which C_B is undefined; if α_i consists of m U-literals, then it corresponds to a transition domain on which C_B is deterministic.

Since in solving the compatibility analysis problem a resolution-based approach is employed, we will state the following proposition concerning the criterion of determinism of a set of clauses using the notion of a resolvent. Let c_1, c_2 be clauses, l be a literal, and $c_1 = c_1' \vee l$ and $c_2 = c_2' \vee \neg l$. The clause $c = c_1' \vee c_2'$ is called a *resolvent* of c_1 and c_2 upon the literal l. Here, a resolvent is meant in the sense of [7], i.e. a resolvent upon a literal of rank 0.

A set of clauses C is said to be *reduced* if it does not contain a clause subsumed by any other clause in C.

Proposition 22. *Let C^* be a maximal reduced set of singular U-clauses not subsumed by clauses in C_B such that all the resolvents of clauses from C^* with clauses from $C_B \cup C^*$ are tautologies. Then the set of clauses $C_B \cup C^*$ is a determinization of C_B.*

Proof. Let $r \geqslant r_B$ be the rank of C^*. We show that $B'(r, C_B \cup C^*)$ is a determinization of $B'(r, C_B)$. Let $F_B^*(t)$ denote the formula represented by $C_B \cup C^*$. Consider the expansions of the formulas $\neg F_B(t)$ and $\neg F_B^*(t)$ consistent with C_B and $C_B \cup C^*$, respectively.

We shall first show that $C_B \cup C^*$ is deterministic. Assume that this is not so. Then according to Proposition 21, in the expansion of $\neg F_B^*(t)$ there exists α_i which contains neither literals for all predicate symbols in U nor a pair of complementary literals. Let l be the literal corresponding to the predicate symbol which does not occur in α_i. Construct the clause c equivalent to the formula $\neg(\varphi_i \& l)$. From the consistency of the expansion with $C_B \cup C^*$ it follows that the resolvent of this clause and any appropriate clause in $C_B \cup C^*$ is the tautology and no clause in $C_B \cup C^*$ subsumes c. This contradicts the maximality of C^*.

We now show that $C_B \cup C^*$ is undefined on those and only those transition domains in $\Sigma(r, F_B(t))$ on which C_B is also undefined. Assume that for some α_i, which in the expansion of $\neg F_B(t)$ does not contain complementary literals, such literals appear in the expansion of $\neg F_B^*(t)$. Then $C_B \cup C^*$ contains a pair of clauses (one of which belongs to C^*) whose resolvent is not the tautology. This contradicts the assumptions about C^*. Consequently, $B'(r, C_B \cup C^*)$ is a determinization of $B'(r, C_B)$. □

The determinizations obtained according to Proposition 22 such that their rank does not exceed the rank of C_B will be called *prime*. With every state space of rank $r \geqslant r_B$ we associate a class of determinizations of C_B denoted by $Det(r, C_B)$. The determinizations of this class are obtained by determinizing C_B on every transition domain in $\Sigma(r, F_B(t))$ (on which C_B is nondeterministic) according to some variant of prime determinizations of C_B. Let C^* be a set of clauses such that $C_B \cup C^*$ is a prime determinization of C_B, and φ be an elementary conjunction defining the domain $N(s, y)$ for some $s \in \Sigma(r, F_B(t))$, $y \in Y$. Then the set of clauses we need to add to C_B to determinize it on the domain $N(s, y)$ according to this variant of prime determinization is of the form $\{\neg\varphi \vee c \mid c \in C^*\}$.

Our consideration so far has been concerned with the determinizations of the automaton $B'(r, C_B)$. It can be shown that if C_B is such that for the corresponding set of normal clauses C'_B any normalized resolvent c (upon a literal of rank 0) of clauses from C'_B is subsumed by a clause in C'_B, then every determinization of $B'(r, C_B)$ uniquely determines the determinization of $B(r, C_B)$. Assuming that the clause set C_B satisfies the above condition, we may regard determinizations of $B'(r, C_B)$ as determinizations of $B(r, C_B)$.

The following proposition holds.

Proposition 23. *For every basic determinization B^* of the r-normal form of automata specified by the set of clauses C_B there exists the determinization in $Det(r+1, C_B)$ which specifies the class of automata equivalent to B^*. Conversely, for every determinization from $Det(r+1, C_B)$ there exists a basic determinization of $B(r, C_B)$ specified by this determinization of C_B.*

This proposition establishes a connection between determinizations from the set $Det(r+1, C_B)$ and basic determinizations of r-normal form of automata specified by the clause set C_B.

5. The resolution-based approach to compatibility analysis of automata

We consider the problem of checking whether a partial, nondeterministic, cyclic $(X-Y)$-automaton A specified by the set of clauses C_A of rank r_A is compatible with a completely defined, nondeterministic, cyclic $(Y-X)$-automaton B specified by the set of clauses C_B of rank r_B.

In this section, we shall show that this problem can be solved by means of the deduction procedure with R-resolution and E-resolution. We now give some definitions.

Definition 24. R-*resolution* (restricted resolution) is an inference rule which only admits resolving upon literals of rank 0.

Definition 25. An R-*deduction* of a clause c from a set of clauses C is a finite sequence of clauses $c_1, \ldots, c_k$ such that $c_k = c$ and each c_i $(i = 1, \ldots, k)$ either belongs to C, or is an R-resolvent of c_j and c_k for $j, k < i$, or is a result of normalization of c_{i-1}.

Definition 26. Let C_E be a set of singular clauses which is called an *eliminating set* and c' be a clause with literals of rank at most -1. The clause c' is *an* E-*resolvent of a normal clause* c *and the eliminating set* C_E if there exist $c_1,\dots,c_n$ such that $c_1=c$, $c_n=c'$ and c_i $(i=2,\dots,n)$ is an R-resolvent of c_{i-1} and some clause in C_E. The rule generating an E-resolvent is called E-resolution.

Application of E-resolution to a clause c corresponds to linear deduction [4] with the top clause c and side clauses in C_E.

Definition 27. An RE-deduction of a clause c from a set of clauses C with an eliminating set C_E is a finite sequence of clauses $c_1,\dots,c_k$ such that $c_k=c$ and each c_i $(i=2,\dots,k)$ either belongs to C, or is an R-resolvent of two preceding clauses, or is an E-resolvent of c_{i-1} and the eliminating set C_E, or is the result of normalization of c_{i-1}.

First, consider the case of deterministic B. Let A^* and B^* be the r-normal forms $(r=\max(r_A,r_B))$ of the automata A and B specified by the sets of clauses C_A and C_B, respectively. Then the subautomaton of A^* formed by the set of all states in A^* that are the reflections of states in B^* is specified by the set of clauses $C_A\cup C_B$. It has been shown in [7] that the automaton $A'(r,C)$ specified by the set of clauses C does not contain a cyclic subautomaton iff there exists an R-deduction of the empty clause $(\Box)$ from C. Thus, Theorem 12 from Section 2 can be restated as follows.

Theorem 28. *The automaton* A *is not compatible with the automaton* B *iff there exists an* R-*deduction of* $\Box$ *from* $C_A\cup C_B$.

Let now B be a *nondeterministic*, completely defined, cyclic $(Y-X)$-automaton. Let C_D be a reduced set of singular U-clauses not subsumed by clauses in C_B and satisfying the following conditions:

(1) the rank of C_D is at most the rank of C_B;
(2) all the resolvents of clauses in C_D with clauses in C_B are tautologies;
(3) for every prime determinization $C_B\cup C^*$ and every $c\in C^*$ there is a clause in C_D which subsumes c.

It can be shown that having the clause set C_D, we can reconstruct the set of all prime determinizations of C_B and hence, the set $Det(r,C_B)$ for any $r\geqslant r_B$. Thus, the set C_D uniquely determines (by Proposition 23) the set of all basic determinizations of any r-normal form of B. There exists a simple algorithm for constructing C_D, but considering it is beyond the scope of the paper.

Theorem 29. *The automaton* A *is not compatible with the automaton* B *if there exists an* RE-*deduction of* $\Box$ *from* $C_A\cup C_B$ *with the eliminating set* C_D.

Proof. We shall show that the existence of the RE-deduction of $\Box$ implies the existence of such a determinization of B specified by the set of clauses $C_B\cup C^*$ that there exists an R-deduction of $\Box$ from $C_A\cup C_B\cup C^*$. We will show how to construct C^*.

Given an RE-deduction of $\square$, let $\varphi_1,\ldots,\varphi_s$ be the sequence of all E-resolvents in this RE-deduction taken in the order they appear in it. For each $i=1,\ldots,s$ we shall construct the set of clauses C_i^* taking $C_0^*=\emptyset$, $C_i^*=C_{i-1}^*\cup C_{\varphi_i}$, where C_{φ_i} is defined as follows. Let $c_1,\ldots,c_q$ be all the side clauses of the linear deduction associated with the E-resolvent φ_i. For $k=1,\ldots,q$ we denote by C_k the set of clauses corresponding to a CNF of the formula $\varphi_i\vee\neg(\psi_1\&\ldots\&\psi_l)\vee x_k$, where x_k is the literal of rank 0 in c_k and ψ_j $(j=1,\ldots,l)$ are all the clauses such that $(\psi_j\vee\neg x_k)\in C_{i-1}^*$ and the resolvent of $\varphi_i\vee x_k$ and $\psi_j\vee\neg x_k$ is not the tautology. If C_{i-1}^* does not contain such clauses, then $C_k=\{\varphi_i\vee x_k\}$. Now define C_{φ_i} as $\bigcup_{k=1}^{q}C_k$. Obviously, C_s^* satisfies all the requirements to a clause set which should be added to C_B to determinize it (Proposition 22) with the possible exception of maximality. In the latter case it can be completed to a maximal one in an arbitrary way.

It remains to show the existence of an R-deduction of $\square$ from $C_A\cup C_B\cup C^*$. We shall construct the R-deduction making use of the RE-deduction of $\square$ from $C_A\cup C_B$ with the eliminating set C_D. The deduction being constructed coincides with the RE-deduction in all points except for the applications of E-resolution. Consider the linear deduction corresponding to an E-resolvent φ. Let $c_1,\ldots,c_k$ be all side clauses of this deduction. When constructing the R-deduction, the E-resolvent φ is replaced with the linear deduction obtained from the linear deduction associated with φ by substituting $\varphi\vee x_i$ for each side clause c_i, where x_i is the literal of rank 0 in c_i. If for some c_i the clause $\varphi\vee x_i$ does not belong to C^*, then this clause is added to the set of clauses from which $\square$ is being deduced. We shall show that adding this clause results in the equivalent set of clauses. Let $\varphi\vee x_i$ be the first such clause added and C' be equal to $C_A\cup C_B\cup C^*$ plus all the clauses which belong to the part of the R-deduction that had been constructed by the moment of adding the clause $\varphi\vee x_i$. We show that C' is equivalent to $C'\&(\varphi\vee x_i)$. Since $(\varphi\vee x_i)\notin C^*$, then C_i associated with c_i in constructing C_φ is a set of clauses corresponding to the formula $\varphi\vee\neg(\psi_1\&\ldots\&\psi_l)\vee x_i$. In this case, C' contains the clauses $(\psi_j\vee\neg x_i)$ $(j=1,\ldots,l)$ such that $\varphi\vee\psi_j\not\equiv 1$. Each of the clauses $\psi_1,\ldots,\psi_l$ coincides with some E-resolvent occurring in the preceding part of the RE-deduction and therefore also in the part of the R-deduction that has been constructed. Thus, C' contains the clauses corresponding to the formula $\varphi\vee\neg(\psi_1\&\ldots\&\psi_l)\vee x_i$ as well as the clauses $\psi_1,\ldots,\psi_l$. The formula $(\varphi\vee\neg(\psi_1\&\ldots\&\psi_l)\vee x_i)\&\psi_1\&\ldots\&\psi_l$ is equivalent to the formula $(\varphi\vee x_i)\&\psi_1\&\ldots\&\psi_l$. Consequently, adding the clause $\varphi\vee x_i$ gives the set of clauses equivalent to C'. This consideration is valid for every subsequent addition of a new clause, because all E-resolvents occurring in the preceding part of the RE-deduction occur in C' obtained by the moment of such an addition. Having replaced all E-resolvents with the corresponding linear deductions as was described above, we obtain an R-deduction of $\square$ from the set of clauses equivalent to $C_A\cup C_B\cup C^*$. As it has been shown in [7], the clause set C is unsatisfiable (in the class of interpretations under consideration) iff there exists an R-deduction of $\square$ from C. Thus, $C_A\cup C_B\cup C^*$ is unsatisfiable, because it is equivalent to an unsatisfiable set and hence there exists an R-deduction of $\square$ from $C_A\cup C_B\cup C^*$. $\square$

The following example illustrates the above proof.

Example 30. The specification of the automaton A is defined by the following set of clauses C_A:

$(w(t-1) \vee u(t) \vee w(t))$ 1,

$(\neg u(t-1) \vee \neg w(t))$ 2,

$(\neg w(t-1) \vee \neg w(t))$ 3,

$(\neg u(t-2) \vee u(t-1) \vee u(t))$ 4,

$(w(t-2) \vee u(t-2) \vee u(t-1) \vee \neg u(t))$ 5.

The set of clauses C_B specifying the automaton B is of the form

$(\neg u(t-1) \vee \neg u(t))$ 6,

$(w(t-1) \vee \neg u(t))$ 7.

It is easy to verify that the clause set

$$\{(\neg u(t))\,\alpha, \quad (u(t-1) \vee \neg w(t-1) \vee u(t))\,\beta\}$$

satisfies all the requirements to the clause set C_D. Let us show the holdness of the least obvious requirement (3) in the definition for C_D. A set C^* corresponding to any prime determinization of C_B consists of singular clauses with a single literal of rank 0 (either $\neg u(t)$ or $u(t)$). Any clause containing the literal $\neg u(t)$ is subsumed by the clause α. Any clause in C_D containing $u(t)$ and not subsumed by clauses in C_B must contain the literals $u(t-1)$ and $\neg w(t-1)$ for its resolvents with clauses in C_B to be tautologies. The only such clause of rank 1 is the clause β.

The RE-deduction of $\square$ from $C_A \cup C_B$ with the eliminating set C_D looks as follows:

$(4,\alpha)$ $(\neg u(t-2) \vee u(t-1))$ $8' \Rightarrow (\neg u(t-1) \vee u(t))$ 8,

$(8,\alpha)$ $(\neg u(t-1))$ $9' \Rightarrow (\neg u(t))$ 9,

$(9,\beta)$ $(u(t-1) \vee \neg w(t-1))$ $10' \Rightarrow (u(t) \vee \neg w(t))$ 10,

$(1,9)$ $(w(t-1) \vee w(t))$ 11,

$(9,10)$ $(\neg w(t))$ 12,

$(11,12)$ $(w(t-1))$ $13' \Rightarrow (w(t))$ 13,

$(12,13)$ $\square$.

All the clauses throughout this example are labelled by numbers or letters (to the right of the clause). The labels of the resolving clauses are indicated in parentheses to the left of the resolvents.

We shall show how to construct an appropriate set C^*.

- E-resolvent $8'$ yields the clause $(\neg u(t-2) \vee u(t-1) \vee \neg u(t))$ a.
- E-resolvent $9'$ yields the clause $(\neg u(t-1) \vee \neg u(t))$ subsumed by the clause 6.
- E-resolvent $10'$ yields $(u(t-1) \vee \neg w(t-1) \vee \neg(\neg u(t-2) \vee u(t-1)) \vee u(t)) = (u(t-2) \vee u(t-1) \vee \neg w(t-1) \vee u(t))$ b.

Thus, C^* consists of two clauses labelled by letters a and b.

Constructing the R-deduction of $\square$.

$(4, a)$ $(\neg u(t-2) \vee u(t-1))$ $8' \Rightarrow (\neg u(t-1) \vee u(t))$ 8,

$(8, 6)$ $(\neg u(t-1))$ $9' \Rightarrow (\neg u(t))$ 9.

Adding the clause $(u(t-1) \vee \neg w(t-1) \vee u(t))$ c.

$(9, c)$ $(u(t-1) \vee \neg w(t-1))$ $10' \Rightarrow (u(t) \vee \neg w(t))$ 10.

The rest of the R-deduction of $\square$ repeats the corresponding part of the RE-deduction.

To prove the converse (to Theorem 29) assertion it will be necessary to consider the transformation of an R-deduction into an RE-deduction. In considering such a transformation, it is convenient to represent an R-deduction in the form of a deduction tree [9]. A deduction tree for an R-deduction is a rooted tree whose nodes are labelled with clauses. Every internal node (a node which is not a leaf node) of this tree has one or two immediate predecessors. If a node has one predecessor it is labelled with the clause which is the result of normalization of the clause labelling its predecessor. If a node has two predecessors (they are called *the counterparts* of each other) then it is labelled with the resolvent of the clauses labelling its predecessors.

A subtree of an R-deduction tree such that all its nodes except the root are labelled with normal clauses and the root is labelled with a clause that contains no literals of rank 0 is called *an eliminating tree*. This is the structural component of an R-deduction tree to which the transformation will be applied. Clearly, that in an eliminating tree every internal node has two immediate predecessors.

Let T be a tree with the set of nodes V and $v \in V$. Denote by T_v a subtree of T with the set of nodes $V_v = \{v' \mid v \text{ is reachable from } v'\}$ (v is reachable from itself), and by $\overline{T}_v$ a subtree of T with the set of nodes $\overline{V}_v = V \setminus V_v \cup \{v\}$.

The following lemma justifies the eliminating trees transformations that will be used below.

Lemma 31. *Let T be an eliminating tree that has the root labelled with φ and C^* be a subset of singular clauses labelling some leaves of this tree. The eliminating tree T can be transformed into the tree T' that satisfies the following conditions*:

(1) *the label φ' of its root subsumes φ ($\varphi' \subseteq \varphi$);*

(2) *all the leaves of T' are the leaves of T;*

(3) *there is a node v in T' such that T'_v does not contain nodes labelled with clauses from C^* and $\overline{T'_v}$ is a deduction tree for a linear deduction with the top clause labelling v and the side clauses in C^*.*

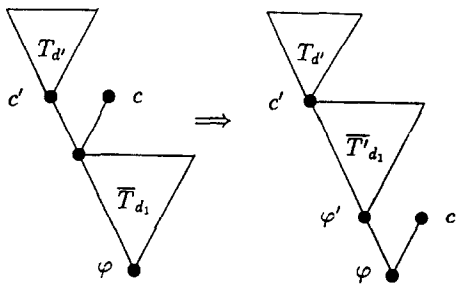

Fig. 4.

Proof. In the eliminating tree T, select a node d' such that its counterpart d is labelled with a clause $c \in C^*$ and $\overline{T}_{d'}$ does not correspond to a linear deduction with side clauses in C^*. Let x be the resolved literal in c. If among the descendants of d' (or d) there are no nodes labelled with clauses containing the literals x or $\neg x$, then rearrange T as follows. Let d' be labelled with c' and c_1 be the resolvent of c and c' labelling the node d_1, the immediate successor of d' and d. Replace the deduction subtree for c_1 with the deduction subtree for c'. After appropriate correction of the resolvents labelling the nodes reachable from d' we obtain the deduction tree for the clause $\varphi' \vee \neg x$, where $\varphi' \subseteq \varphi$. Then, resolving $\varphi' \vee \neg x$ and c we get φ. Thus, the resolution with the clause $c \in C^*$ (labelling the node d) can be moved immediately to the root which gives the tree with the same root's label φ. This transformation of T is pictured in Fig. 4. Let among the descendants of d' be a node labelled with a clause containing the literal $\neg x$. In this case, resolving with c is redundant and it may be removed from the deduction of φ. The tree corresponding to the resulting deduction is a deduction tree of the clause $\varphi' \subseteq \varphi$.

Now consider the case where among the descendants of d' is a node labelled with a clause containing the literal x. Select such a node closest to the root of T and replace the deduction subtree for the clause labelling this node with d. After appropriate correction of the deduction, we obtain the deduction tree of the clause $\varphi' \subseteq \varphi$. Since the selected node is closest to the root of T, among the descendants of d in the resulting tree, there are no nodes labelled with clauses containing the literal x. Therefore this tree can be transformed as described above.

Using this technique, we can transform every eliminating tree to the intended form. □

In an eliminating tree T' satisfying the condition (3) of Lemma 31, we will distinguish two parts: the R-part corresponding to R-deduction subtree T'_v and the E-part corresponding to $\overline{T'_v}$.

Theorem 32. *If the automaton A is not compatible with the automaton B, then there exists an* RE*-deduction of $\square$ from $C_A \cup C_B$ with the eliminating set C_D.*

Proof. If A is not compatible with B, then there exists (by Theorems 13 and 28) a determinization $C_B \cup C^*$ of the clause set C_B, such that $\square$ is R-deducible from $C_A \cup C_B \cup C^*$. Let T be a deduction tree for an R-deduction of $\square$ from $C_A \cup C_B \cup C^*$. If we replace any eliminating subtree of T with its transformation according to Lemma 31 and correspondingly adjust the rest of T (if φ' is not identical with φ), we obtain the new R-deduction tree for $\square$.

By iterating this process of transformation we obtain an R-deduction in which every eliminating tree satisfies the conditions in Lemma 31. Having replaced an E-part of every such subtree with the application of the E-resolution rule, we obtain an RE-deduction of $\square$ from $C_A \cup C_B$ with the eliminating set C^*. By the definition of C_D, for each clause $c \in C^*$ there exists a clause in C_D that subsumes c. Therefore, for every E-resolvent φ of any clause and the eliminating set C^* there exists the corresponding E-resolvent φ' of the same clause and the eliminating set C_D such that $\varphi' \subseteq \varphi$. This implies the existence of an RE-deduction of $\square$ from $C_A \cup C_B$ with the eliminating set C_D. $\square$

Theorems 29 and 32 give the method to check if the specification of an automaton is compatible with the specification of its environment. The procedure of generating new clauses by application of R-resolution, RE-resolution, and normalization terminates if either the empty clause is obtained, which indicates that the specification of the automaton is not compatible with the environment specification, or the application of these rules yields no new clauses. In the latter case the specifications of the automaton and its environment are compatible. Employing the deletion strategy [4] enhances the efficiency of the procedure. A slight modification of this procedure makes it possible to solve the more general task, namely, to distinguish in the set of all automata satisfying the specification a subset of automata compatible with the environment. The specification of this subset represented as a specification of a nondeterministic automaton can be extracted from the set of clauses obtained after the termination of the modified deduction procedure.

6. Conclusion

The notion of compatibility of automata was introduced in [5] in order to formalize the requirements which should be satisfied by interacting automata. The problem of compatibility analysis of automata arises in the development of a system that interacts with its environment. Under the assumption made for the automaton that models the environment (it is assumed to be completely defined), partiality of the automaton to be designed is the only source of its possible incompatibility with the environment automaton. When declarative specification is used, we can never decide in advance (not constructing the corresponding automaton) if the specified automaton is partial or not. Therefore, any specification of the automaton under design must be checked for compatibility with the environment specification. The results presented in this paper

made it possible to develop an efficient resolution-based procedure for checking the compatibility of an automaton specification with the specification of its environment. This procedure is implemented in the framework of the system for automated design of automata from logical specifications. Efficiency of this procedure is determined by the following features of the method.

First, we manage to reduce the amount of required determinizations of the environment specification (prime determinizations of C_B are considered instead of the set $Det(r+1, C_B)$). Secondly, these determinizations are not handled separately, i.e. the compatibility analysis is not made for every prime determinization, instead, they are specified by the clause set C_D and the compatibility analysis is accomplished as a single procedure analogous to that for deterministic automata. The latter is a deduction procedure and its efficiency, in turn, is based on using restricted resolution [7] as a deduction rule.

Acknowledgements

The authors thank the anonymous referees for helpful comments and constructive criticism that led to improvements in the presentation of the paper.

References

[1] M. Abadi, L. Lamport, The existence of refinement mapping, Theoret. Comput. Sci. 82 (1991) 253–284.
[2] M. Abadi, L. Lamport, Composing specifications, ACM Trans. Programming Languages Systems 15 (1993) 73–132.
[3] D. Brand, P. Zafiropulo, On communicating finite-state machines, J. ACM 30 (1983) 323–342.
[4] C.L. Chang, R.C.T. Lee, Symbolic Logic and Mechanical Theorem Proving, Academic Press, New York, 1973.
[5] A.N. Chebotarev, Interacting automata, Cybernet. Systems Anal. (translated from Russian) 27 (1991) 810–819.
[6] A.N. Chebotarev, Determinations of logical specifications of automata, Cybernet. Systems Anal. (translated from Russian) 31 (1995) 1–7.
[7] A.N. Chebotarev, M.K. Morokhovets, Consistency checking of automata functional specifications, in: Proc. LPAR'93, Lecture Notes in Artificial Intelligence, vol. 698, Springer, Berlin, 1993, pp. 76–85.
[8] W. Chen, J.T. Udding, T. Verhoeff, Networks of communicating processes and their (de-)composition, in: Mathematics of Program Construction, Lecture Notes in Computer Science, vol. 375, Springer, Berlin, 1989, pp. 174–196.
[9] R. Cleaveland, B. Steffen, When is "partial" adequate? A logic-based proof techique using partial specifications, in: Proc. 5th Ann. Symp. on Logic in Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 440–449.
[10] A. Gill, Introduction to the Theory of Finite-State Machines, McGraw-Hill, New York, 1962.
[11] O. Grumberg, D.E. Long, Model checking and modular verification, ACM Trans. Programming Languages Systems 16 (1994) 843–871.
[12] D. Harel, A. Pnueli, On the development of reactive systems, in: K.R. Apt (Ed.), Logics and Models of Concurrent Systems, NATO ASI Series, vol. F13, Springer, Berlin, 1985, pp. 477–498.
[13] S.S. Lam, A.U. Shankar, Protocol verification via projections, IEEE Trans. Software Eng. 10 (1984) 325–342.

[14] N.A. Lynch, M.R. Tuttle, Hierarchical correctness proofs for distributed algorithms, in: Proc. 6th Symp. on the Principles of Distributed Computing, ACM, New York, 1987, pp. 137–151.

[15] A. Pnueli, Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends, in: J.W. de Bakker, W.P. de Roever, G. Rozenberg (Eds.), Current Trends in Concurrency, Lecture Notes in Computer Science, vol. 224, Springer, Berlin, 1986, pp. 510–584.