Modeling and Checking Networks of Communicating Real-Time Processes[1]

Jürgen Ruf and Thomas Kropf
Institute of Computerdesign and Fault Tolerance (Prof. D. Schmid)
University of Karlsruhe, Kaiserstr. 12, Geb. 20.20, 76128 Karlsruhe, Germany
{ruf,kropf}@ira.uka.de
http://goethe.ira.uka.de/hvg/cats/raven

Abstract. In this paper we present a new modeling formalism that is well suited for modeling real-time systems in different application areas and on various levels of abstraction. These *I/O-interval structures* extend interval structures by a new communication method, where input sensitive transitions are introduced. The transitions can be labeled time intervals as well as with communication variables. For interval structures, efficient model checking techniques based on MTBDDs exist. Thus, after composing networks of I/O-interval structures, efficient model checking of interval structures is applicable. The usefulness of the new approach is demonstrated by various real-world case studies, including experimental results.

1 Introduction

For modeling real-time systems it is necessary to have a formalism which allows the explicit declaration of timing information, e.g., a valve needs 5 seconds to open. Moreover, for real-time systems, typical delay times may vary, e.g., due to fabrication tolerances and thus have to be represented by time intervals indicating minimal and maximal time bounds. Since systems are often described in a modular manner, these modules have to communicate with each other. This means, the modeling formalism must be strong enough for expressing communication.

Many approaches for modeling real-time systems exist. Two main approaches have to be distinguished: those based on timed automata [1] and those extending finite state machines (FSM). Both classes of formalisms have disadvantages.

Timed automata have complex model checking algorithms [2,3,4]. Moreover, timed automata use an event-based communication, i.e., transitions in the automata are labeled with events, and transitions with same labels in different automata are synchronized during composition. This event-based communication is not the usual form of hardware communication. In hardware there exist signals which have a defined value for all time instances and modules use the output signals of other modules as inputs.

FSMs have no possibility to model quantitative timing effects. The expansions of FSMs for modeling real-time systems [5,6] have no intuitive semantics according to the specification logic [7] and there exists no composition strategy which is able to combine many communicating modules.

In [8] we have presented a new formalism called interval structure. This formalism has a proper semantics with regard to the specification logic (CCTL) and it allows the

1.This work is sponsored by the German Research Grant (DFG)

L. Pierre and T. Kropf (Ed.): CHARME'99, LNCS 1703, pp. 265-279, 1999.
© Springer-Verlag Berlin Heidelberg 1999

declaration of time-intervals. We developed an efficient model checking algorithm based on an MTBDD representation of interval structures [9,10]. In [11] we have presented a method for the composition of interval structures completely working on the MTBDD representation, including two minimization heuristics.

A natural model of a hardware system uses a description of several submodules connected by wires. The description of submodules may contain free input variables. If we now want to use interval structures for the modeling of timed sub-systems, the advantages of a compact time representation gets lost, because at any time instance the structure has to react to the free inputs. This effect destroys all timed edges and splits them into unit delay edges. The resulting interval structure is in principal an FSM with many new intermediate states.

Therefore, in this paper we extend interval structures by a signal-based communication mechanism. In order to model this communication properly, the I/O-interval structures contain additional input signals and support input restrictions on timed transitions. With this concept there exist unconditional transitions only consuming time (input-insensitive) as well as conditional transitions which consume time if they are taken because inputs fulfil a input restriction (input-sensitive). With this formalism, timed transitions are even possible if free input variables are introduced.

Furthermore, this communication allows the integration of untimed FSMs (e.g. a controller) and I/O-interval structures (e.g. the timed environment). With some few changes, the efficient composition algorithms of interval structures [11] are applicable to I/O-interval structures. After composing networks of I/O-interval structures, the efficient model checking algorithms of interval structures are applicable [8].

The following section introduces interval structures and their expansion, called I/O-interval structures. In Section 3 the specification logic CCTL is presented. Section 4 gives a short overview about composition and model checking. Some case studies, which show the flexibility of our new approach will be presented in Section 5. We show that for many real-world examples, one clock per module is sufficient and realistic. In Section 6, we show some experimental results. The last section concludes this paper.

2 Using Structures for Modeling System Behavior

In the following sections we introduce Kripke structures and interval structures [8]. In order to overcome some problems of interval structures that result whenever free inputs are introduced, we extend these structures to I/O-interval structures.

2.1 Interval Structures

Structures are state-transition systems modeling HW- or SW-systems. The fundamental structure is the Kripke structure (unit-delay structure, temporal structure) which may be derived from FSMs.

Definition 2.1. *A Kripke structure (KS) is a tuple* $U = (P, S, T, L)$ *with a finite set* P *of atomic propositions.* S *is a finite set of states,* $T \quad S \times S$ *is the transition relation connecting states. We assume that for every state* $s \quad S$ *there exists a state* $s' \quad S$ *such that* $(s, s') \quad T$*. The labeling function* $L \colon S \qquad (P)$ *assigns a set of atomic propositions to every state.*

The semantics of a KS is defined by paths representing computation sequences.

Definition 2.2. *Given a KS* $U = (P, S, T, L)$ *and a starting state* $s_0 \quad S$. *A path is an infinite sequence of states* $p = (s_0, s_1, \ldots)$ *with* $(s_i, s_{i+1}) \quad T$.

The basic models for real-time systems are interval structures, i.e., state transition systems with additional labelled transitions. We assume that each interval structure has exactly one clock for measuring time. The clock is reset to zero if a state is entered. A state may be left if the actual clock value corresponds to a delay time labelled at an outgoing transition. The state must be left if the maximal delay time of all outgoing transitions is reached (Fig. 2.1).

Definition 2.3. *An interval structure (IS) is a tuple* $= (P, S, T, L, I)$ *with a set of atomic propositions* P, *a set of states* S *(i-states), a transition relation between the states* $T \quad S \times S$ *such that every state in* S *has a successor state, a state labeling function* $L{:}S \quad (P)$ *and a transition labeling function* $I{:}T \quad (I\!N)$ *with* $I\!N = \{1, \ldots\}$.

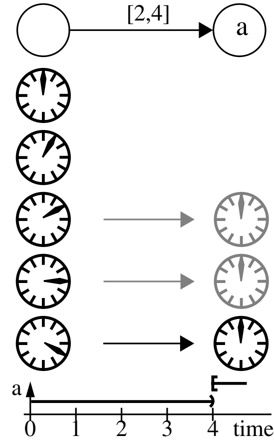

Fig. 2.1. Example IS

The only difference to KSs are the transitions which are labeled with delay times. Every state of the IS must be left after the *maximal state time*.

Definition 2.4. *The maximal state time of a state* s $\mathrm{MaxTime}{:}S \quad I\!N_0$ *(*$I\!N_0 = \{0, \ldots\}$*) is the maximal delay time of all outgoing transitions of* s, *i.e.*

$$\mathrm{MaxTime}(s) = max\{t \mid \quad s'.(s, s') \quad T \quad t = max(I(s, s'))\} \quad . \tag{1}$$

Besides the states, we now also have to consider the currently elapsed time to determine the transition behavior of the system. Hence, the actual state of a system, called the *generalized state*, is given by an i-state s and the actual clock value v.

Definition 2.5. *A generalized state (g-state)* $g = (s, v)$ *is an i-state* s *associated with a clock value* v. *The set of all g-states in an IS* $= (P, S, T, L, I)$ *is given by:*

$$G = \{(s, v) \mid s \quad S \quad 0 \quad v < \mathrm{MaxTime}(s)\} \tag{2}$$

The semantics of ISs is given by runs which are the counterparts of paths in KSs.

Definition 2.6. *Given the IS* $= (P, S, T, L, I)$ *and a starting state* g_0. *A run is a sequence of g-states* $r = (g_0, g_1, \ldots)$. *For the sequence holds* $g_j = (s_j, v_j) \quad G$ *and for all j it holds either*

- $g_{j+1} = (s_j, v_j + 1)$ *with* $v_j + 1 < \mathrm{MaxTime}(s_j)$ *or*
- $g_{j+1} = (s_{j+1}, 0)$ *with* $(s_j, s_{j+1}) \quad T$ *and* $v_j + 1 \quad I(s_j, s_{j+1})$.

The semantics of an IS may also be given in terms of KSs. Therefore in [11] we defined a stutter state expansion operation on ISs, which transforms an IS into a KS. Fig. 2.2 shows an example of expanding timed transitions with stutter states.

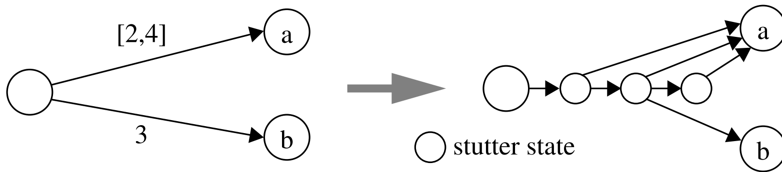

Fig. 2.2. Expansion Semantics of a Timed Structure using Stutter States

IS have no explicit input variables, i.e., if different ISs communicate with each other, they have to share state variables. This communication scheme leads to the following problems:

- one transition may depend on many different valuations of the input variables
- there may exist timed transitions which do not depend on (free) inputs but on the other hand, a variable in an IS may not be declared as don't care.

2.2 Interval Structures with Communication

To understand the second problem, we consider a transition with delay time four in a IS, which is not dependent on an input variable (i). It is however a member of the set of atomic propositions P of the IS. As defined in Def. 2.3, IS are not expressive enough to represent this transition, except by splitting this transition into many unit delay edges as shown in Fig. 2.3. After this splitting operation, the transition may be taken, regardless of the input i, but the advantage of the compact time representation of IS gets lost.

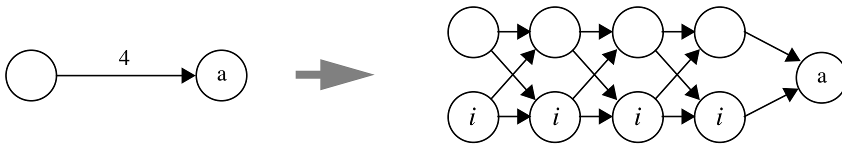

Fig. 2.3. Modeling a free input i with an IS

To overcome these problems which become worse during composition, we introduce a new modeling formalism: I/O-interval structures. These structures carry additional input labels on each transition. Such an input label is a Boolean formula over the inputs. We interpret this formulas as input conditions which have to hold during the corresponding transition times. Input-insensitive edges carry the formula $true$. The formula $false$ should not exist, since this transition can never be selected.

In the following definition we formalize Boolean formulas with sets of valuations over the input variables. An element of the set $Inp := \wp(P_I)$ defines exactly one valuation of the input variables: the propositions contained in the set are true, all others are false. An element of the set $\wp(Inp)$ then defines all possible input valuations for one edge. For example, given the inputs a and b, the set $\{\{a\}, \{a, b\}\}$ is represented by the Boolean function $(a \wedge \neg b) \vee (a \wedge b) = a$. This example shows that the variable b does not affect the formula, i.e., the transition labeled with the formula a may be taken independent of the input b.

Definition 2.7. *An I/O-IS is a tuple* $_{I/O} = (P, P_I, S, T, L, I, I_I)$. *For accessing the first (second) component of an element* $x \quad S \times Inp$ *we write:* $x[1]$ $(x[2])$. *(This access operator is defined to all elements consisting of multiple components)*

- *The components* P, S, L *and* I *are defined analogously to IS*
- P_I *is a finite set of atomic input propositions*
- *The transition relation connects pairs of states and inputs:* $T \quad S \times S \times Inp$
- $I_I : T \qquad (Inp)$ *is a transition input labeling function*

We assume the following restriction on the input labelling:

$$t_1 \quad T. \; t_2 \quad T. \begin{array}{c} (t_1[1] = t_2[1]) \quad (t_1 \quad t_2) \\ (I_I(t_1) = I_I(t_2)) \quad (I_I(t_1) \quad I_I(t_2) = \quad) \end{array} \tag{3}$$

This formula ensures that if there exist multiple edges starting in the same, then their input restrictions are either equal or disjoint. This means, that the input valuations on timed edges are clustered.

Definition 2.8. *The cluster function* $C : S \times Inp \qquad (Inp)$ *computes all input valuation of a cluster represented by an arbitrary member*

$$C(s, i) := \begin{cases} I_I(t) & \text{if} \quad s' \quad S. \; i' \quad Inp. t = (s, s', i') \quad T \quad i \quad I_I(t) \\ & \text{otherwise} \end{cases} \tag{4}$$

All input evaluations belonging to the state are clustered. Because of equation (3) all clusters of one state are disjoint, i.e. every evaluation of the input variables represents the cluster it lies in.

Now we describe the semantics of the I/O-ISs by defining runs. Therefore we first need the maximal state time.

Definition 2.9. *The maximal state time* MaxTime:$S \times Inp \qquad I\!N$ *is the maximal delay time of all outgoing transitions, i.e.*

$$\begin{array}{l} \text{MaxTime}(s, i) := \\ max\{v \mid \; s' \quad S. \; i' \quad Inp. t = (s, s', i') \quad T \quad i \quad I_I(t) \quad v = max(I(t))\} \end{array} \tag{5}$$

G-states in I/O-IS also have to consider the actual inputs besides the i-state and the elapsed time.

Definition 2.10. *An extended generalized state (xg-state)* $g = (s, i, v)$ *is an i-state* s *associated with an input evaluation* $i \quad Inp$ *and a clock value* v. *The set of all xg-states in an I/O-IS is given by:*

$$G_I = \left\{ (s, i, v) \mid s \quad S \quad i \quad \bigcup_{i' \quad Inp} C(s, i') \quad 0 \quad v < \text{MaxTime}(s, i) \right\} \tag{6}$$

Definition 2.11. *Given the I/O-IS* $\quad = (P, P_I, S, T, L, I, I_I)$, *a run is a sequence of xg-states* $r = (g_0, g_1, \ldots)$. *For the sequence holds* $g_j = (s_j, i_j, v_j) \quad G_I$ *and for all j it holds either*

- $g_{j+1} = (s_j, i_{j+1}, v_j + 1)$ *with* $v_j + 1 < \mathrm{MaxTime}(s_j, i_j)$ *and* $i_{j+1} \quad C(s_j, i_j)$ *or*
- $g_{j+1} = (s_{j+1}, i_{j+1}, 0)$ *with* $t = (s_j, s_{j+1}, i_{j+1}) \quad T, i_j \quad I_I(t)$ *and* $v_j + 1 \quad I(t)$.

I/O-IS may also be expanded to KS. The expansion works similar to IS, except for input sensitive transitions. These transitions are expanded as shown in Fig. 2.3. After the expansion, the efficient composition and model checking algorithms for ISs are applicable.

Since we want to examine reactive systems of connected I/O-ISs, we assume that for every xg-state and for every input evaluation there exists a successor xg-state. This means for edges with delay times greater one there has to be a fail state in which is visited if the actual input does not fulfill the input restriction. This implies that transitions either have no input restriction or, if a transition with delay time ∍ has an input restriction, there must exist a transition with interval $[1, ∍ - 1]$ which connects the same starting state with the fail state. For unit-delay edges we have to ensure that for all input evaluations there exists a successor state. In general, the I/O-interval structures are not restricted to reactive systems, but the examples studied later should be reactive.

For an easier understanding of the following case studies, we introduce a graphical notation for I/O-IS in Fig. 2.4. The a_i represent the inputs of the modeled sub-system, and f denotes a function $f : P_I^n \quad I\!B$.

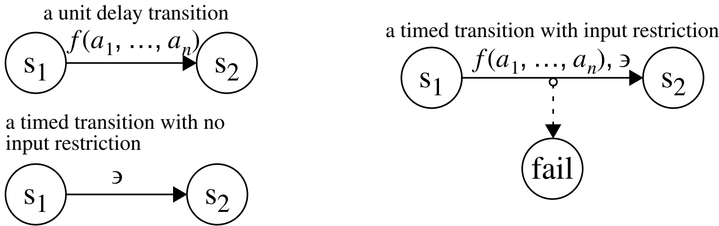

Fig. 2.4. Graphical notations

3 Specifying Real-Time Properties with CCTL

CCTL [11] is a temporal logic extending CTL with quantitative bounded temporal operators. Two new temporal operators are introduced to make the specification of timed properties easier. The syntax of CCTL is the following:

$$
\begin{aligned}
&p \mid \neg(\mid (\quad (\mid (\quad (\mid (\quad (\mid (\;) \quad (\\
(:=& \mid \mathsf{EX}_{[a]}(\mid \mathsf{EF}_{[a,b]}(\mid \mathsf{EG}_{[a,b]}(\mid \mathsf{E}((\;\mathsf{U}_{[a,b]}(\;) \mid \mathsf{E}((\;\mathsf{C}_{[a]}(\;) \mid \mathsf{E}((\;\mathsf{S}_{[a]}(\;) \qquad (7)\\
&\mid \mathsf{AX}_{[a]}(\mid \mathsf{AF}_{[a,b]}(\mid \mathsf{AG}_{[a,b]}(\mid \mathsf{A}((\;\mathsf{U}_{[a,b]}(\;) \mid \mathsf{A}((\;\mathsf{C}_{[a]}(\;) \mid \mathsf{A}((\;\mathsf{S}_{[a]}(\;)
\end{aligned}
$$

where $p \quad P$ is an atomic proposition and $a \quad I\!N$ and $b \quad I\!N \# \{*\}$ are time-bounds. All interval operators can also be accompanied by a single time-bound only. In this case the lower bound is set to zero by default. If no interval is specified, the lower bound is implicitly set to zero and the upper bound is set to infinity. If the X-operator has no time bound, it is implicitly set to one. The semantics of the logic is given as a validation relation:

Definition 3.1. Given an IS $\quad = (P, S, T, L, I)$ and a configuration $g_0 = (s, v) \quad G$.

$$
\begin{aligned}
g_0 \models p \quad &:\Leftrightarrow \quad p \quad L(s) \\
g_0 \models \neg (\quad &:\Leftrightarrow \quad \text{not } g_0 \models (\\
g_0 \models ((\quad -) \quad &:\Leftrightarrow \quad g_0 \models (\text{ and } g_0 \models -
\end{aligned}
\tag{8}
$$

$$
\begin{aligned}
g_0 \models \mathsf{EG}_{[a,b]}(\quad &:\Leftrightarrow \quad \text{there ex. a run } r = (g_0, g_1, \ldots) \text{ s.t.} \\
&\quad\quad \text{for all } a \quad i \quad b \text{ holds } g_i \models (\\
g_0 \models \mathsf{E}((\ \mathsf{U}_{[a,b]} -) \quad &:\Leftrightarrow \quad \text{there ex. a run } r = (g_0, g_1, \ldots) \text{ and an } a \quad i \quad b \text{ s.t.} \\
&\quad\quad g_i \models - \text{ and for all } j < i \text{ holds } g_j \models (\\
g_0 \models \mathsf{E}((\ \mathsf{C}_{[a]} -) \quad &:\Leftrightarrow \quad \text{there ex. a run } r = (g_0, g_1, \ldots) \text{ s.t.} \\
&\quad\quad \text{if for all } i < a \text{ holds } g_i \models (\text{ then } g_a \models - \\
g_0 \models \mathsf{E}((\ \mathsf{S}_{[a]} -) \quad &:\Leftrightarrow \quad \text{there ex. a run } r = (g_0, g_1, \ldots) \text{ s.t.} \\
&\quad\quad \text{for all } i < a \text{ holds } g_i \models (\text{ and } g_a \models -
\end{aligned}
\tag{9}
$$

The other operators may be derived by the defined ones, e.g.:

$$
\begin{aligned}
\mathsf{EF}_{[a,b]}(\quad &. \quad \mathsf{E}(true \ \mathsf{U}_{[a,b]}(\ ) \\
A((\ \mathsf{C}_{[a]} -) \quad &. \quad \neg \mathsf{E}((\ \mathsf{S}_{[a]} \neg -) \\
A((\ \mathsf{S}_{[a]} -) \quad &. \quad \neg \mathsf{E}((\ \mathsf{C}_{[a]} \neg -)
\end{aligned}
\tag{10}
$$

Examples for the use of CCTL may be found in the case studies.

4 Composition and Model Checking

Model checking as described in [8] works on exactly one IS, but real-life systems are usually described modular by many interacting components. In order to make model checking applicable to networks of communicating components, it is necessary to compute the product structure of all submodules. This product-computation is called composition.

Due to space limitations we do not describe the composition algorithms in detail. The principal idea is to define a reduction operation ($reduce$) on KS which replaces adjacent stutter states by timed edges. With this operation we are able to define the composition by:

$$
{}_1 \parallel {}_2 := reduce(expand({}_1) \parallel expand({}_2))
\tag{11}
$$

The algorithms described in [11] works as follows:

1. expand all modules and substitute their input variables by the connected outputs
2. compose the expanded structures
3. reduce the composition KS

In order to avoid the computation of the complete unit-delay composition structure, Step 2 and Step 3 may be performed simultaneously by incrementally adding only the reachable transitions.

Since we adapt the expansion to I/O-ISs, the composition algorithms are also applicable if we work with closed systems, where every free variable is bounded by a module. After composition, the efficient model checking algorithms for IS may be used for verification. These algorithms are described in detail in [8]. Composition as well as the model checking algorithms use the symbolic representation of state sets and transition relations with MTBDDs during computation.

5 Case Studies

In this section, we show the flexibility of our approach to model real-time systems. We have chosen three systems out of very different areas and on different levels of abstraction. This section shows, that we do not necessarily need timed automata to represent complex real-time systems.

5.1 Modeling Hardware with Time Delays

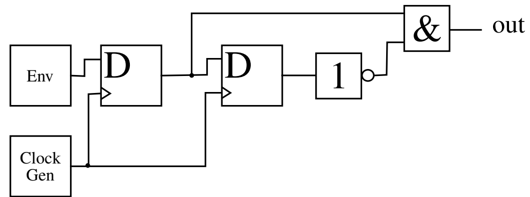

Fig. 5.1. The circuit of the single-pulser

A first class of real-time systems are digital hardware circuits with delay times. These circuits are composed by digital gates like AND gates or flipflops, but they have a specific timing behavior. Impulses on the inputs which are shorter than the delay time of a gate are suppressed. Only if an impulse stays constant at least for the delay time, the gate may change its outputs. For modeling flipflops, we assume a setup and a hold time. If the input signals violate these time-constraints, the flipflop remains in its actual state. As an easy example we choose the single-pulser circuit [12] which is shown in Fig. 5.1. Figure 5.2 shows the basic gates (initial states are bold). For modeling the AND gate, input sensitive edges have been used: If the system starts in the state *high*, and the inputs fulfil $\neg a_1 \quad \neg a_2$ for $\ni$ time units then it changes to the state *low*. If the inputs fulfill $a_1 \quad a_2$ before $\ni$ time units are passed, the structure remains in the state *high*. Here, I/

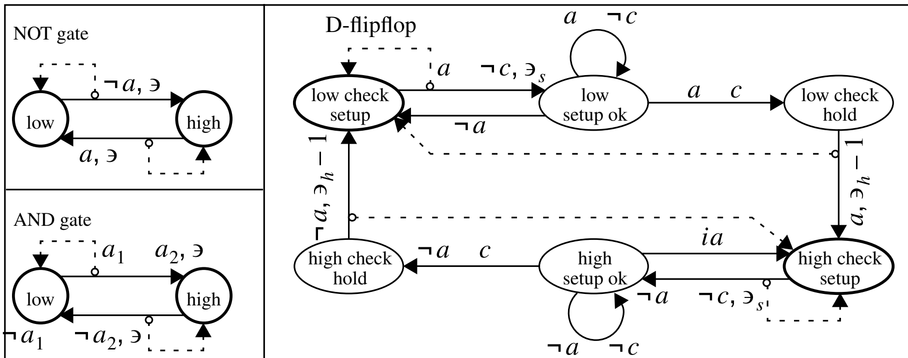

Fig. 5.2. Basic gates with input i, i_1, i_2, clock c and delay time $\ni$ resp. setup $\ni_s$ and hold $\ni_h$

O-IS allow a very clear and compact modeling of timing constraints and communication behavior.

The clock generator is modeled by an I/O-IS with two states (*high* and *low*) which toggles between both states every cycle-time. The environment is either a human pressing the button (which should be single pulsed) or it's a bounding pulse.

The specification checks that an output signal appears if the input stays high long enough:

$$spec1 := \mathsf{AG}(\neg Env.out \quad \mathsf{EX}(\mathsf{A}(Env.out\ \mathsf{C}_{[2\ni_c + \ni_h]}\mathsf{AF}_{[\ni_a - \ni_h]}And.out))) \qquad (12)$$

The following specification verifies that the output stays high for one cycle period and then changes to low for a further cycle and afterwards rests low until the input becomes high:

$$spec2 := \mathsf{AG}(\neg And.out \quad \mathsf{EX}(And.out \quad \mathsf{A}(And.out\ \mathsf{C}_{[2\ni_c + \ni_n]}tmp1)))$$
$$tmp1 := \mathsf{AG}_{[2\ni_c - 1]}\neg And.out \quad \mathsf{A}((\neg And.out)\mathsf{U}Env.out) \qquad (13)$$

5.2 Modeling a Production Automation Systems

The production cell is a case study to evaluate different formal methods for verification [13]. A schematic deposition of the production cell is shown in Fig. 5.3. A feed belt moves work pieces to the elevation rotary table. This table lifts the pieces to the robot arm 1. The robot arm 1 moves the work pieces to the press, where robot arm 2 removes them after the work has performed. Robot arm 2 drops the work pieces over the deposit belt. The robot has two arms to gain a maximal performance of the press.

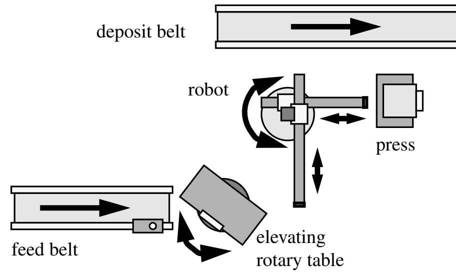

Fig. 5.3. Schematic disposition of the production cell

Modeling the feed belt (FB, Fig. 5.4): The FB delivers the cell with work pieces (wp) every $\ni_{min}$ or $\ni_{max}$ seconds (state feed). If the wp is moved to the end of the belt (state work piece present) the belt stops rotating until the *next* signal raises. Then it shifts the wp to the elevating rotary table. If the table is not in the right position, the wp falls to the ground or blocks the table (state fail). This action takes $\ni_{reload}$ seconds for completion

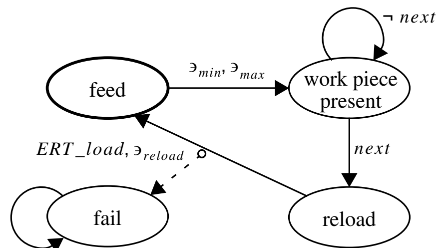

Fig. 5.4. I/O-IS modeling the feed belt

Modeling the elevating rotary table (ERT, Fig. 5.5): The ERT has a belt for loading wps, which starts rotating by a rising *load* signal. The *mov* signal makes the ERT moving and rotating from the low loading position to the high unloading position (and

vice versa). The signal *pick* (*drop*) indicates that a wp is picked up (dropped down) by the robot arm 1. The signal *pick* is a combination of many signals, the right robot rotary table position, the out position of the arm, the empty signal of the arm and the magnet on signal of the controller.

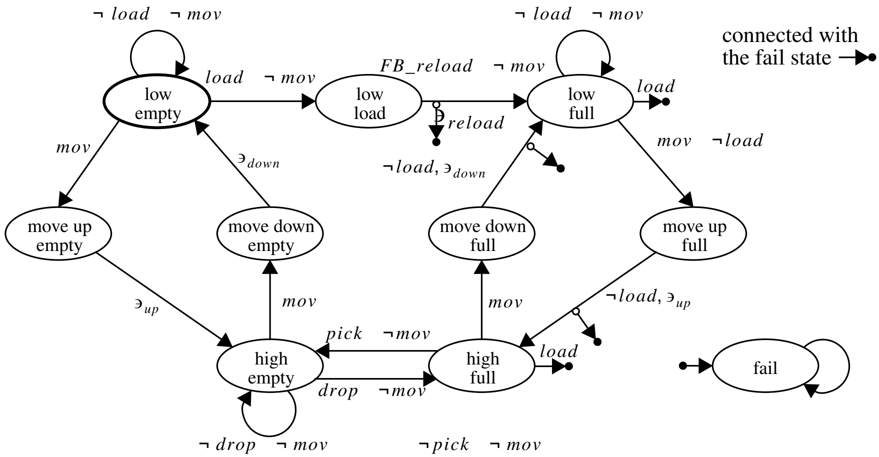

Fig. 5.5. I/O-IS modeling the elevation rotary table

The **robot** is modeled by three separate modules, the two **arms (RA, Fig. 5.6)** and the **robot rotary table (RRT, Fig. 5.7)**. A RA may be empty or full (a wp is hanging on the RA) and it may be extended (out) or retracted (in). The fail state indicates that the wp falls to the ground. The RRT has three positions: unload ERT (UE) unload press (UP) and drop wp (DW). Every movement between these positions is possible.

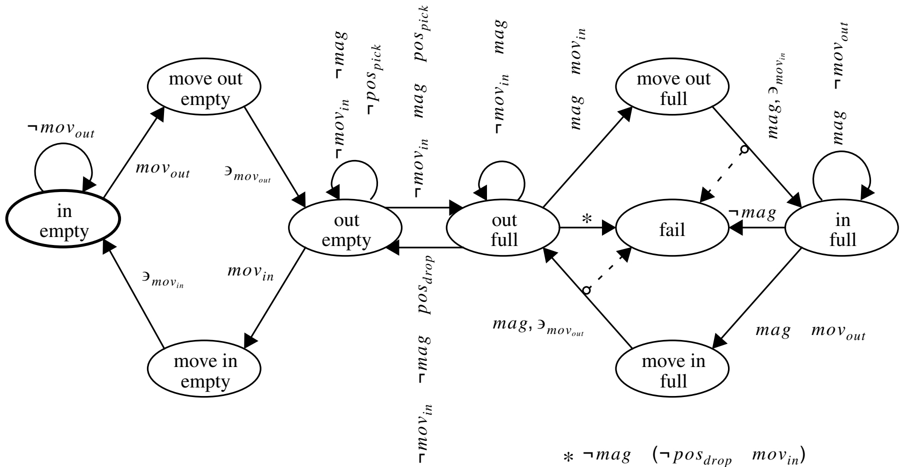

Fig. 5.6. I/O-IS modeling the robot arm

Modeling the press (P, Fig. 5.8): The P has three positions: low for unloading wps by RA2, mid for loading wps by RA1 and press. The signal *lift* (*lower*) makes the press

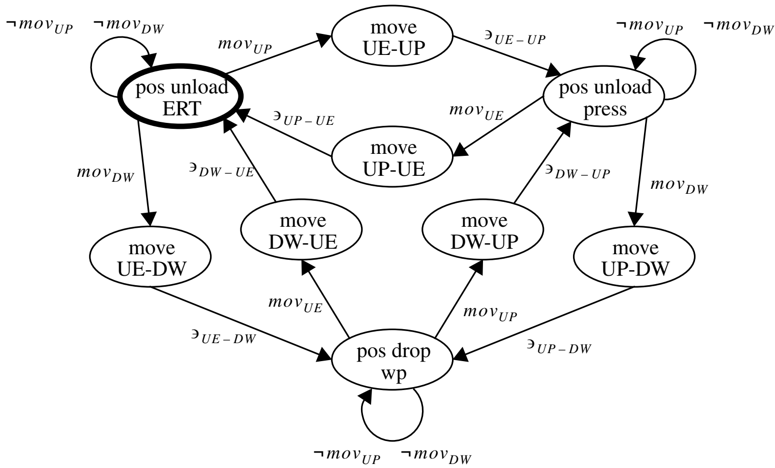

Fig. 5.7. I/O-IS modeling the robot rotary table

to move to the next higher (lower) position. The *drop* and *pick* signals are connected with the RAs and the RRT.

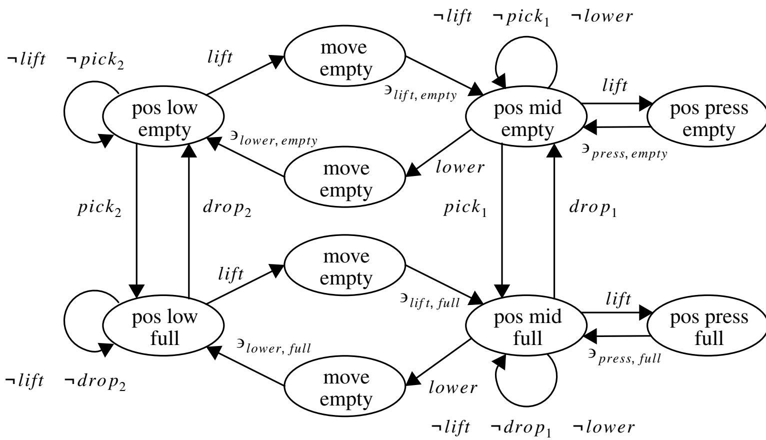

Fig. 5.8. I/O-IS modeling the press

The deposit belt is not modeled by a separate module, it is assumed that the belt is always running. If the RA 2 drops a wp over the deposit belt, it is transported.

The controlling unit is a synchronous FSM described with an output and a transition function. We assume a sampling rate of 5ms for the external signals. The controller manages the correct interaction between all modules.

The checked specifications check time bounds for the arrival of the first pressed wp, or the cycle time of pressed wps. It is also shown that 4 wps may stay in the production cell, but not five. Also some wait times for the wps are shown.

5.3 Modeling a Bus Protocol

The next example is the arbitration mechanism of a bus protocol. We modeled the J1850 protocol arbitration [14] which is used in on- and off-road vehicles. The protocol is a CSMA/CR protocol. Every node listens to the bus before sending (carrier sense, CS). If the bus is free for a certain amount of time, the node starts sending. It may happen that two or more nodes simultaneously start sending (multiple access, MA). Therefore, while sending, every node listens to the bus and compares the received signals to the send signals. If they divide, it looses arbitration (collision resolution, CR) and waits until the bus is free again. A sender distinguishes between two sending modes, a passive and an active mode. Active signals override passive signals on the bus. Succeeding bits are alternately send active and passive. The bits to be send are encoded by a variable pulse width: a passive zero has a pulse width of 64/ sec , a passive one bit takes 128/ sec , an active zero bit takes 128/ sec and an active one bit takes 64/ sec . The bus is simply the union of all actively send signals. The arbitration is a bit-by-bit arbitration, since a (passive/active) zero shadows a one bit. Before sending the first bit, the nodes send an SOF (start of frame) signal, which is active and takes 200/ sec . In Fig. 5.9 some examples of arbitration are shown. We assume an exact frame length of 8 bits. After sending the last bit, the sender sends a passive signal of 280/ sec , the end of frame (EOF) signal.

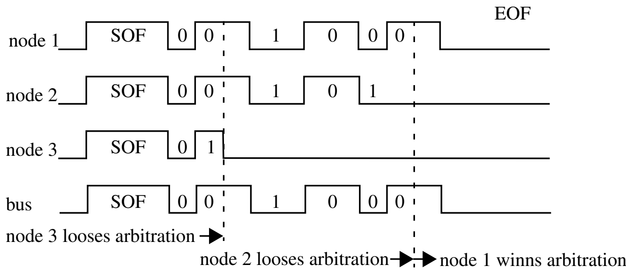

Fig. 5.9. Some examples of arbitration

One bus node is modeled by two sub-modules: a sender/receiver and a counter. Initially, all modules are in their initial states. If the node decides to send (indeterministically) the sender/receiver listens to the bus. If the bus stays low for $\ni_{CS}$ time units, the module changes to the SOF state. The counter is triggered by the continue high/low states of the sender. In the initial state, the counter module sends the *count* signal. After sending the SOF signal, the sender sends alternately passive and active one and zero bits. If the bus becomes active while sending a passive bit, the sender/receiver changes to the CS state and tries sending again later.

6 Experimental Results

In this section we compare the MTBDD model checking approach against an ROBDD approach which uses the complete expanded unit-delay transition relation for verification. Both algorithms prove the same formulas, only the representation and the model checking techniques are different. For the ROBDD approach we use our unit-delay model checker which uses the same BDD package as the MTBDD approach.

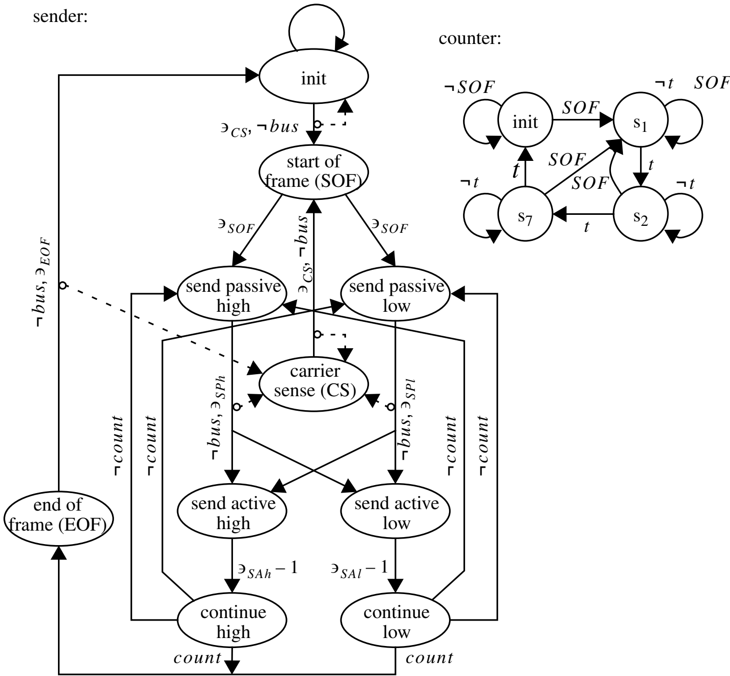

Fig. 5.10. Two submodules modeling one bus node

Both algorithms start with the same description of the networks of I/O-ISs and expand these descriptions to unit-delay structures. The MTBDD approach then performs the IS composition [11]. Afterwards it uses the algorithms presented in [8] for model checking. The ROBDD approach preforms standard product composition on Kripke structures and then checks the resulting Kripke structure against the same specifications.

Table 1 shows the number of ROBDD respective MTBDD nodes for the three case studies presented in section 5. The J1850 example contains three bus nodes.The max delay column indicates the maximal delay times used in the models. The ROBDD column shows the number of nodes necessary for the corresponding Kripke structure. The MTBDD column is divided into three sub columns, the first one shows the number of MTBDD nodes computed by the composition algorithm, the second and the third column shows the number of MTBDD nodes after applying a minimization algorithm (reencoding, minimal path) presented in [11]. The reencode algorithm reduces the number of MTBDD variables encoding the clock values (introduced by the expansion step) by substituting these encodings by new ones with a minimal number of encoding variables. The minpath algorithm adds unreachable states to the state space in the MTBDD representation to shrink the number of MTBDD nodes from the root to the leaves. Both algorithms do not affect the behavior of the system.

Table 1. BDD nodes comparison

	max delay	ROBDD nodes	MTBDD nodes		
			no min.	reenc.	minp.
single pulser	500	27,691	4,176	1,546	1,917
production cell	4,000	1,972,238	69,003	-[a]	18,999
J1850	300	350,358	76,610	389,478	50,301

a. memory exceeded 500MB due to a bad variable ordering required for this algorithm

Table 2 compares the run-times and the memory usage of the MTBDD approach against the ROBDD approach.

Table 2. run-time and memory usage comparison

	complete	composition	minimization	model checking	memory (MByte)
single-pulser					
MTBDD-approach	15.77	14.83	0.43	0.46	4.59
ROBDD-approach	11.59	8.84	-	2.56	5.23
production cell					
MTBDD-approach	39:38	38:01	00:18	01:17	19.22
ROBDD-approach	54:43	27:07	-	27:20	177.6
J1850 bus arbitration					
MTBDD-approach	05:06	00:37	00:09	04:19	22.2
ROBDD-approach	248:28	01:07	-	247:19	78.2

7 Conclusion

In this paper we have presented a new modeling formalism (I/O-interval structure) which is well suited for many real-time systems of different application areas and abstraction levels. The formalism extends interval structures by a proper communication method based on signal communication between modules. This technique overcomes the problem of splited edges appearing in interval structures while introducing free input variables.

On the other hand, this signal-based communication allows a natural modeling of hardware systems, and allows an easy combination of timed systems and synchronous controllers. This formalism leads to a compact model representation of real-time systems with MTBDDs. Moreover, after composing I/O-interval structures, the efficient model checking algorithms working on MTBDDs are applicable to the modeled systems.

References

[1] R. Alur and D. Dill. Automata for Modeling Real-Time Systems. In *International Colloquium on Automata, Languages and Programming*, *LNCS*, NY, 1990. Springer-Verlag.

[2] R. Alur, C. Courcoubetics, and D. Dill. Model Checking for Real-Time Systems. In *Symposium on Logic in Computer Science*, Washington, D.C., 1990. IEEE Computer Society Press.

[3] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic Model Checking for Real-Time Systems. In *IEEE Symposium on Logic in Computer Science (LICS)*, pages 394–406, Santa-Cruz, California, June 1992. IEEE Computer Scienty Press.

[4] M. Bozga, O. Maler, A. Pnueli, and S. Yovine. Some progress in the symbolic verification of timed automata. In O. Grumberg, editor, *Conference on Computer Aided Verification (CAV)*, volume 1254 of *LNCS*, pages 179–190. Springer Verlag, June 1997.

[5] S. Campos and E. Clarke. Real-Time Symbolic Model Checking for Discrete Time Models. In T. Rus and C. Rattray, editors, *Theories and Experiences for Real-Time System Develpment*, AMAST Series in Computing. AMAST Series in Computing, May 1994.

[6] J. Frößl, J. Gerlach, and T. Kropf. An Efficient Algorithm for Real-Time Model Checking. In *European Design and Test Conference (EDTC)*, pages 15–21, Paris, France, March 1996. IEEE Computer Society Press (Los Alamitos, California).

[7] J. Ruf and T. Kropf. Using MTBDDs for discrete timed symbolic model checking. *Multiple-Valued Logic – An International Journal*, 1998. Special Issue on Decision Diagrams, Gordon and Breach.

[8] J. Ruf and T. Kropf. Symbolic model checking for a discrete clocked temporal logic with intervals. In *Conference on Correct Hardware Design and Verification Methods (CHARME)*, pages 146–166, Montreal, Canada, October 1997. Chapman and Hall.

[9] R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic Decision Diagrams and Their Applications. In *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 188–191, Santa Clara, California, November 1993. ACM/IEEE, IEEE Computer Society Press.

[10] E. Clarke, K. McMillian, X. Zhao, M. Fujita, and J.-Y. Yang. Spectral Transforms for large Boolean Functions with Application to Technologie Mapping. In *ACM/IEEE Design Automation Conference (DAC)*, pages 54–60, Dallas, TX, June 1993.

[11] J. Ruf and T. Kropf. Using MTBDDs for composition and model checking of real-time systems. In *FMCAD 1998*. Springer, November 1998.

[12] S. Johnson, P. Miner, and A. Camilleri. Studies of the single pulser in various reasoning systems. In *International Conference on Theorem Provers in Circuit Design (TPCD)*, volume 901 of *LNCS*, Bad Herrenalb, Germany, September 1994. Springer-Verlag, 1995.

[13] C. Lewerentz and T. Lindner, editors. *Formal Development of Reactive Systems - Case Study Production Cell*, number 891 in LNCS. Springer, 1995.

[14] SAE. J1850 class B data communication network interface. *The Engeneering Society For Advancing Mobility Land Sea Air and Space*, October 1995.