Modeling a Production Cell Component as a Hybrid Automaton: A Case Study*

Heinrich Rust**

Lehrstuhl für Software Systemtechnik, BTU Cottbus

Abstract. HyTech, a system to model and analyse linear hybrid systems, is used to model a belt component of a production cell. A strategy is demonstrated for building a model for both the physical components of the belt and its control program, and for proving some properties about the system modeled. On the basis of the experiences of the case study, several concepts for hybrid specification languages are developed which would allow a more convenient modeling of hybrid systems.

1 Introduction

The ability to analyse hybrid systems is becoming more important with the increasing use of computers in control applications. In a lot of applications, during the analysis of the system it is possible to abstract from the continuous character of important quantities of the systems. Looking at the total development process of control software for reactive systems, these approaches have to deal with a special problem: There is one more step in the validation process for the system models, because it must be checked if the abstraction step from the hybrid system to the purely discrete system is correct.

This extra check is made easier if the abstraction step is small. Hybrid description notations thus might help to check correspondence of a model and the modeled system.

There are different types of notations for hybrid systems. Notations for timed systems use a restricted form of continuously varying variables: These always change with the same rate as the time. In this paper, we will use linear hybrid automata: The change of continuously varying variables can be described by piecewise constant derivatives.

There are several tools for modeling and proving of properties of timed and hybrid systems, with several different approaches for the tasks of a user. Deductive provers, e.g. based on PVS [ORSvH95], have the function of a proof checker. The proofs can be constructed by a human, but the easier steps and the checks for correctness and completeness are performed with the help of a program. Other tools are based on automatized state space exploration. Examples are UppAal [BLL+96], Kronos [DOTY96] and HyTech [HHWT95]. Other tools try to combine the approaches (e.g. STeP [BBC+96]).

* Submitted for presentation at FTRTFT'98

** Reachable at: BTU, Postfach 101344, D-03013 Cottbus, Germany; Tel. +49(355)69-3803, Fax.:-3810; rust@informatik.tu-cottbus.de

In this paper, we will report experiences from an experiment using HyTech to model and analyse a hybrid system. HyTech has been selected because it allows to model a system as a combination of linear hybrid atomata, not just timed automata. See [ACH+95] for an introduction to the theory underlying the analysis of linear hybrid automata.

We do not emphasize the mathematical problems which are associated with the application of HyTech hybrid automata to the modeling and analysis of hybrid systems. We are more concerned with the usability of the notation and the possibilities to give structure to a specification of a hybrid system.

The way to checkable system descriptions is via redundancy which can be checked for consistency, if possible automatically. If bits of information which are intended to describe the same system are found to be inconsistent, this points out misunderstandings or modeling errors. This idea is simple. The difficult part is to design a notational system which both makes it attractive for the modeler to input redundant information, and which allows different kinds of misunderstandings and errors to become explicit in this way. We believe that HyTech is a good base on which one could build to develop such a notation.

The context of this work is the development of practically usable methods for development and verification of control software for reactive systems. The final goal is to develop a notation for the control process which allows both execution as a control program and formal analysis of a system. In this way, we want to support the what-you-verify-is-what-you-run principle.

As field of application, we choose a component of a production cell. In [LL95], a case study is described which has been used in a lot of different approaches to model a reactive system and prove critical properties. In this work, we will try to extend the existing approaches in the direction of modeling time in a quantitative way.

We will present a model for a component of a production cell, not for a whole cell. This component is a transport belt. We model both important physical aspects of the belt, the control software, and the context of the belt.

The rest of the paper has the following structure. First, we present the relevant properties of the transport belt we are going to model. We first describe some physical properties, then the control program. Afterwards, we give a short introduction to the concepts used in HyTech for modeling and analysing hybrid systems. Section 4 and Section 5 are the main sections of this study. There, we describe our HyTech-model of the transport belt and some possible verifications, and we present our experiences while we implemented and verified the model.

2 A transport belt

2.1 The physical model

We built a physical model of a production cell for experiments in the practical application of formal methods to the development of control software for reactive systems. Our production cell contains several different types of components. Some of these components occur several times. There are transport belts with no, one and two sensors. There

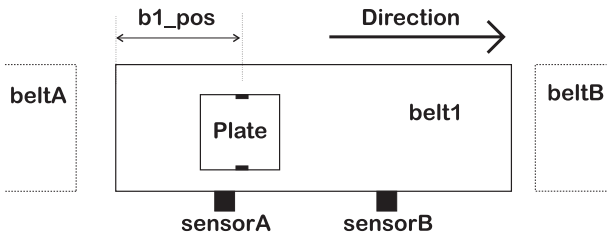

Fig. 1. The belt

are turning belts and pushers, there is a repository for the plates to be worked on, and there are several other component types. These components have different kinds of sensors and actuators.

For this case study, we selected to model a transport belt with two sensors. It is controlled via two bit-valued inputs: One determines if the belt is in motion or not, and the other determines the direction of the belt motion. In this study, we will consider a control program in which one of the moving directions will not be used. Thus, we can consider the belt as either moving or stopped.

The transport belt has two magnet sensors which sense the presence of a plate in some range around their positions. The plates which travel in our system are small wood blocks which have screws in the middles of their sides. These screws can be sensed when they are in some range around the magnet sensors.

The longitudinal position of the plate on the belt and the position and the sensing radius of the sensor are not the only parameters which determine if a sensor senses a plate or not. Another is the lateral position of the plate on the belt. There are some tolerances for this lateral position. We will not model this lateral position explicitly, but we will use nondeterminism to model the differences in longitudinal positions which lead to a plate being sensed or not.

In addition to the position of the sensors on the belt, there are other static parameters: its length and the speed when switched on. This is the speed with which a plate on the belt is moved. We will abstract from acceleration and deceleration when the belt is started or stopped.

Via its two endpoints, the belt can receive a plate from one neighbouring belt and transfer it to the other one. We will refer to the endpoint from which plates can be loaded as to endpoint `A`, and to the other one as to endpoint `B`, and we will refer to the neighbouring belts as `beltA` and `beltB`.

To the sensor near endpoint `A`, we will refer to as `sensorA`, and to the other as `sensorB`. We assume that `sensorA` is far enough from endpoint `A` so that when it senses a plate, it is ensured that this plate has totally left the neighbour at endpoint `A`. For `sensorB`, we assume that as long as it senses a plate, this has not yet left the belt. In Figure 1, we display a drawing of the belt.

2.2 The control program

The control program for the belt which we will consider has to do the following:

- At the beginning, assume that the belt is empty and stopped.
- Wait till the neighbour at endpoint `A` wants to transfer a plate. In some way, we have to model this communication between different components.
- Start belt motion. Wait till the plate arrives at `sensorA`. Tell `neighbourA` that the plate has been received.
- Move the plate till it reaches `sensorB`. Stop the belt. Wait till `neighbourB` is ready to receive the plate.
- When `neighbourB` has accepted to receive the plate, start the belt again to transfer the plate to the neighbour.
- Wait until `neighbourB` tells that the plate has been transfered. Then stop the belt.
- Start again at the beginning.

3 HyTech

HyTech [HHWT95] is a tool for modeling hybrid systems and analysing the system for reachability properties, developed at Cornell University. There have been several generations of the tool. For the work described in this paper, we used version 1.04. We will describe only those properties of HyTech of which we made use.

In HyTech, the hybrid system is described as the parallel composition of a set of automata with finite control. There is a globally visible set of continuously changing variables. Some of these can be declared to have a restricted set of time derivatives, but we will not use this feature. The parallel automata communicate via these variables and via CSP-style synchronization labels. We will sometimes use the expression 'signal' for a synchronization label, especially when we want to emphasize that we often can consider, via a semantical interpretation, a label to be generated by one component automaton and received by one or several others.

3.1 The locations of an automaton

Each automaton consists of a finite set of locations, a finite set of transitions, and a finite set of synchronization labels. One of the locations is the initial location of the automaton. With each location, two pieces of information are associated: There is an invariant which describes the values the global variables are allowed to have when the location is active, and there is a set of restrictions for the time-derivatives of global variables. The restrictions for the time-derivates define the allowed development of the analogue component of the state space. The invariant can be used to force transitions to be taken after some time.

3.2 The transitions of an automaton

Each transition of the automaton leads from one location to another. With each transition, a guard is associated. This guard is, like the invariant, a restriction for the global variables. It expresses a condition which must be fulfilled for the transition to be taken.

Another piece of information which might be associated with a transition is a set of nondeterministic assignments to global variables. This allows to express noncontinuous changes of the values of the global variables.

A synchronization label might be associated with a transition. These labels have a CSP-like semantics: A transition with a synchronization label s can only be taken when in all parallel automata having s in their alphabet, a transition is taken which has s as synchronization label. Thus, multi-way synchronization of discrete transitions in parallel automata can be accomplished.

Finally, a transition can be marked as urgent. This means that after time has passed in the location from which the transition departs, this transition can not be used.

3.3 The product automaton

HyTech builds a product automaton from the parallely declared automata. The resulting automaton has the same elements as the component automata which are described above. The set of locations of the product automaton is a subset of the cartesian product of the location sets of the component automata.

Invariants of product states and guards of product transitions which arise from common synchronization labels are constructed as intersections of the invariants rsp. guards of the locations or transitions combined.

3.4 The form of invariants, guards and derivation restrictions

The form of the invariants, guards and derivation restrictions is the key to automatic checkability of properties of the described system. If we simplify a bit, we can say that allowed are finite conjunctions of inequalities of linear expressions over the set of global variables for invariants and guards, and finite conjunctions of inequalities of linear expressions of the time-derivatives of the global variables for the derivation restrictions. These representations of configuration sets of the global variables or their derivations allow efficient set-theoretic operations.

We explain the notation when it is first used.

3.5 Analysis of the described system

HyTech uses a special kind of programming language for the analysis of a system which is described by a hybrid automaton. This programming languages contains the following components:

– Variables can be declared which contain regions. Regions are sets of configurations the hybrid automaton might be in. One configuration consists of two components. The first is a discrete component which is an element of the cartesian product of the location sets of the component automata. The second is a possibly continuous component. This is an assignment of real numbers to all the continuous variables of the automaton.
– There are several types of expressions. Some of them are: Constant expressions for regions, region variables, and the application of operators to region expressions. These operators include set-theoretic operations, the one-step computation of the region reachable forward or backward from a given region, and the computation of the region reachable from a given region in any number of steps. Existential quantification can remove restrictions regarding specific continuous variables or regarding some or all locations from the region description.
  Boolean expressions can also be formulated as boolean combinations of set-theoretic predicates over the regions.
– Statements of the language are assignments of region expression to region variables, printing statements for strings and regions, and a while- and an if-statement.

4 Modeling the transport belt with HyTech

In this section, we describe our approach to model the belt with HyTech. Where appropriate, we give a short conclusion about experiences made while modeling specific aspects of the belt.

We use the following notation in our model: `belt1` and, abbreviated, `b1` are used for the modeled belt. `beltA` and `bA` are used for the belt at endpoint A, and `beltB` rsp. `bB` are used for the belt at endpoint B.

4.1 Global structure

We decided to structure the specification of the system into several communicating automata with different tasks:

– The control program of `belt1` is relatively simple. We will use one component automaton to model it.
– The physical context of `belt1` consists of `beltA` and `beltB`, and both of these have their own control programs. We model these physical neighbours and their control programs very abstractly with just one component automaton.
– The physical model of `belt1` is represented by three component automata. We use one automaton to model the speed of a transported plate. This automaton is also used to model the loading and unloading of a plate from the belt. Further, we use one automaton to model each of the sensors.
– We designed a set of testing automata to check some correctness conditions of our physical model.

```
-- Constants of the belt model.
define(b1_length, 80)
define(b1_speed,  1)
define(b1_sensApos, 30)
define(b1_sensBpos, 50)
define(sensor_radius_min, 3)
define(sensor_radius_max, 5)
define(plate_radius, 5)
define(distance_bA_b1, 1)
```

Fig. 2. Constants of the belt model

Experiences: HyTech allowed this decomposition into several distinct automata via the possibility to define parallely executed automata which communicate with each other via the shared global variables and via synchronization labels of the transitions. We only felt that a possibility was missing to express that some of the component automata belonged more nearly to each other, like the components of the physical model or the test automata.

4.2 Constants

In Figure 2, we give the start of the specification, containing the constants used for modeling static physical properties of the model. Lines starting with two dashes are comments. Constants are modeled via the m4-text-replacement mechanism. We use several symbolic constants describing static physical properties of the model:

- `b1_length` and `b1_speed` represent the length of the belt in length units and the speed, when switched on, in length units per time units. Positive speed means that the belt moves from endpoint `A` in the direction of endpoint `B`.
- `b1_sensApos` and `b1_sensBpos` represent the positions of the centers of the sensors on the tape.
- `sensor_radius_max` and `sensor_radius_min` are the sensor radii. If the plate position is inside the minimal sensor radius around a sensor, the sensor must be on. If the plate position is outside the maximal sensor radius around a sensor, the sensor must be off. Between the inner and the outer radius of the sensor, there is the transition from the off- to the on-state on the side towards endpoint `A` and the transition from the on- to the off-state towards endpoint `B`, but where this transitions happens is not determined.
- The physical extension of the plate is described by the constant `plate_radius`. Whenever the distance of the plate's center to an endpoint is less than this, we do not consider the plate to be totally controlled by the belt. E.g., when the distance of the plate to endpoint `B` becomes equal or smaller to this constant, the physical model of the belt initiates the physical transfer of the plate to the neighbour by sending a message.
- The physical distance between the neighbour at endpoint `A` and the belt is modeled by the constant `distance_bA_b1`. This constant is used to compute the initial

```
var
    b1_pos: analog;
    test1_stopped_time: analog;
    test1_bA_b1_crit_time: analog;
    test1_b1_bB_crit_time: analog;
```

Fig. 3. Declaration of the analogue variables

position of the plate relative to the modeled belt when the neighbour at endpoint `A` notices that the plate is starting to leave.

4.3 Modeling the position of a plate on the belt

Figure 3 shows the declaration of the analogue variables we use in our model. The position of the plate on the belt is the only analogue quantity we model explicitly in our model. For this, we use the variable `b1_pos`. The other three variables are used in the definition of a test automaton which is used to ensure correctness conditions. These will be discussed in Section 4.8.

`b1_pos` is the position of the center of the transported plate in relation to endpoint `A` of the belt. This is the endpoint at which the plate are transfered to the belt. A position of 0 thus means that the center of the plate is located at endpoint `A`, and since the plate has a radius of `plate_radius`, it projects from the belt this much.

The variable `b1_pos` will always be a real. Thus we have to solve the problem how to model an unloaded belt. We will use the values of `b1_pos` which are at greater than or equal to `b1_length-plate_radius` for this.

Another question is how to model the transition of a plate from the neighbour at endpoint `A` to the belt and from the belt to the neighbour at endpoint `B`. The difficult phase is when the plate starts to leave one belt and to enter another. There will be some time in which the movement of the plate is determined totally from the movement of the first belt. Then there will be some time in which the movement of the plate can only be determined from the movements of both cooperating belts, and after this, only the movement of the second belt is relevant for the movement of the plate. The problem occurs in the medium phase: When does it start and end? How do we model the movement of the plate for the case that one of the belts stands still and the other moves?

The physical modeling of this behaviour might be very complicated. Thus, we avoid to model this by using the knowledge that such a situation would indicate a mistake of the control program or a breakdown of the system. Instead of building a complete model of the physical representation, we restrict ourselves to the smaller set of situations which are allowed by a correct control program. This means, of course, that we should check that situations for which movement of the plate is not modeled correctly are not reachable under the control of the given control program.

We demonstrate our approach with the transfer of a plate from the modeled belt to the neighbour at endpoint `B`. When we recognize that the plate is starting to project from the belt at endpoint `B`, we send a signal informing the physical model of the neighbour

at endpoint B about this. This is the moment from which we consider the movement of the plate to be dependent on the movement of the neighbour at B, even if in the physical world, this would be the case only later. The speed of the physical plate would nevertheless be modeled correctly by the speed of the modeled plate if both belts stay moving until the plate has totally reached the neighbour at B. Thus, we have to prove that in all situations where we are not sure by which of two belts a plate is controlled, both belts are moving with the same speed. In our case study, this is the transfer of a plate from the neighbour at A to the belt and the transfer from the belt to the neighbour at B.

The start of such a the critical time interval is marked by a signal. The neighbour at A sends the signal sl_bA_to_b1 to inform the belt about the start of the transfer of the plate, and the belt sends the signal sl_b1_to_bB to inform the neighbour at B when the transfer starts between these to components.

The end of the critical time interval is marked by another signal: sl_stop_transfer_bA_b1 for the transfer from the neighbour at A to the belt and sl_stop_transfer_b1_bB for the transfer from the belt to the neighbour at B. These signals are generated by the controller program of the receiving component when it recognizes that the plate has been received.

To prove this correctnes constraint, we thus have to show that the belt is moving from the moment in which sl_bA_to_b1 is received until sl_stop_transfer_bA_b1 is generated, and that it is moving from the moment sl_b1_to_bB is generated until sl_stop_transfer_b1_bB is received. A testing automaton can check these conditions. We will describe this automaton later in Section 4.8.

There is a problem in our model. It is built into the environment automaton that only one plate can be on the belt at any one moment. We could also have proved that the control program ensures this by allowing the environment automaton to try to initiate another transfer of a plate to the belt before the other plate has left the plate, and extending the test automaton to check that there is never more than one plate transfered to the belt by the physical model.

Experiences: We learned how to avoid the modeling of difficult physical configurations by using information about the control program. For this, we developed a technique using test automata to check if our assumptions about the interaction of the control program on one hand and of the physical model on the other hand are correct.

4.4 Modeling the speed of the plate on the belt

In Figure 4, we give a part of the description of the automaton describing movement of a plate on the belt. Three dots (. . .) are used where we left out some lines. The automaton description starts with the keyword automaton and ends with the keyword end. The name of the automaton is b1_movement. After the keyword synclabs, all synchronization labels are listed on which this automaton synchronizes with other automata. This means that a transition of another automaton involving one of these synchronization labels can only be done when this automaton also performs a transition labeled with this synchronization label.

```
automaton b1_movement
    synclabs:
        sl_b1_go, sl_b1_stop, sl_bA_to_b1, sl_b1_to_bB;
    initially loc_stopped_and_unloaded;
    loc loc_stopped_and_unloaded:
        while True wait {db1_pos = 0}
        when True sync sl_b1_go goto loc_moving_and_unloaded;
        when True sync sl_bA_to_b1
            do { b1_pos' = - distance_bA_b1 - plate_radius }
            goto loc_stopped_and_loaded;
    loc loc_moving_and_unloaded:
        while True wait {db1_pos = 0}
        when True sync sl_b1_stop
            goto loc_stopped_and_unloaded;
        when True sync sl_bA_to_b1
            do { b1_pos' = - distance_bA_b1 - plate_radius }
            goto loc_moving_and_loaded;
...
    loc loc_moving_and_loaded:
        while b1_pos <= b1_length-plate_radius
            wait {db1_pos = b1_speed}
        when True sync sl_b1_stop
            goto loc_stopped_and_loaded;
        when b1_pos = b1_length-plate_radius
            sync sl_b1_to_bB goto loc_moving_and_unloaded;
end
```

Fig. 4. Automaton describing movement of a plate

Even if the underlying semantics does not make this distinction, is it helpful to differentiate between input and output labels of an automaton. If we wanted to be exact, we would have to say that input labels are those to which the automaton can react in any configuration, output labels are all other synchronization labels of the automaton. We will use a slightly different definition for input labels: These are labels which do not restrict any other transition to be taken **in the given system**. The first definition is equivalent to saying that input labels do not restrict any other transition in **any** system the automaton is a component of.

According to our definition, the synchronization labels `sl_b1_go`, `sl_b1_stop` and `sl_bA_to_b1` are considered to be generated somewhere else, i.e. they are input labels, while the synchronization label `sl_b1_to_bB` is considered to be generated by the automaton at hand, so this is considered to be an output label.

The keyword `initially` defines the initial configuration of the automaton. Here, a location of the automaton and, optionally, a convex predicate over the global variables can be given. We define that this automaton starts in the location `loc_stopped_and_unloaded`.

The movement of a plate is modeled with four locations. These allow to model all states in which the belt is moving or is stopped, and in which the belt is loaded or is unloaded.

The notation used for locations is the following: Each location description starts with the keyword `loc`. Then follows the name of the location. The keyword `while` introduces the invariant and, after the `wait`-keyword, restrictions for derivatives of the global variables are given. After this `while-wait`-clause, the transitions are described. Each transition starts with the keyword `when`. After this, a guard is given. `True` stands for the trivial guard which is always fulfilled. The `sync`-clause, if present, demands synchronization of the transition with the given synchronization label. The final `goto`-clause gives the location into which the transition leads.

We use this automaton to control the value of the variable `b1_pos`. HyTech allows us to set derivatives and discrete transitions for a given global variable in several different component automata, but it enhances readability to not use this feature.

We have to define the derivative of `b1_pos` in every location of the component automaton. This is done in the `wait`-clauses. Here, we use simple equalities, where the name of the variable is prefixed with the letter `d`. In all but one locations, the time derivative is zero. Only in the location modeling the situation where the belt is both moving and loaded, the time derivative of the position is set to `b1_speed`.

The synchronization labels `sl_b1_stop` and `sl_b1_go` are used to switch between the locations representing movement and their counterparts representing no movement of the belt. When `sl_bA_to_b1` is received, the automaton switches from an unloaded location to a loaded location, since this signal signifies that the neighbour at `A` starts transfer of a plate to this piece. At the same time, `b1_pos` is initialized to `-plate_radius-distance_bA_b1`, as discussed in the previous section.

When the position exceeds `b1_length-plate_radius`, we consider the plate in transition to the neighbour at endpoint `B`. This has been described in the previous section. Here, it is also possible to see a difference between typical input labels and typical output labels: Whereas all transitions for input labels of the automaton had a guard `True`, the transition with the output label has a nontrivial guard.

Experiences: HyTech's synchronization labels were useful to model communication between component automata. It might have been even more convenient to have a less abstract communication mechanism: We would have liked to be able to distinguish input and output signals. Normally, it is not difficult to define if a signal is to be considered as input or output signal, and this information

- might help the reader a lot to interprete the meaning of a given synchronization label, and
- is some easily supplied further redundant information for which consistancy checks might be checked automatically.

In the example, the variable `b1_pos` was only controlled by the given automaton. Every restriction of this variable in another component automaton would have been unintentional. This special relation between a variable and a component automaton should be expressible to make automatic consistency checks possible.

```
automaton b1_sensA
    synclabs:
        sl_b1_sAon, sl_b1_sAoff, sl_bA_to_b1;
    initially loc_off_after;
    loc loc_off_before:
        while b1_pos <= b1_sensApos-sensor_radius_min wait {}
        when b1_pos >= b1_sensApos-sensor_radius_max
            sync sl_b1_sAon goto loc_on;
    loc loc_on:
        while b1_pos <= b1_sensApos+sensor_radius_max wait {}
        when b1_pos >= b1_sensApos+sensor_radius_min
            sync sl_b1_sAoff goto loc_off_after;
    loc loc_off_after:
        while True wait {}
        when True sync sl_bA_to_b1 goto loc_off_before;
end
```

Fig. 5. Automaton describing outputs of sensorA

4.5 Modeling sensor outputs

The automaton in Figure 5 models the output of `sensorA`. It generates two output signals which can be used in automata which have to read the sensors, and it reacts to the loading signal.

We model the sensor with three locations: One of these locations represents the situation in which the sensor is switched on. This location is called `loc_on`. The other two represent situations in which the sensor is switched off. `loc_off_after` represents the situation where there either is no plate on the belt, or the plate has already passed the sensor. `loc_off_before` represents the situation where there is a plate which has not yet reached the sensor. The label `sl_bA_to_b1` which signals the loading of a plate to the belt triggers the transition from `loc_off_after` to `loc_off_before`.

We already remarked that we wanted to model the uncertainty when the sensor switches on for a given plate with two sensibility radii: `sensor_radius_min` is the radius around the sensor position in which a plate is necessarily sensed. `sensor_radius_max` is the radius around the sensor position out of which the plate is never sensed. At a distance between `sensor_radius_min` and `sensor_radius_max`, the transition must take place.

We model this behaviour in the following way: When we are in the location `loc_off_before`, we can stay there as long as the plate did not enter the inner radius. This is expressed in the invariant `b1_pos <= b1_sensApos-sensor_radius_min`. But: We may already leave this location when the plate has entered the outer radius. This is expressed in the guard of the transition to the location `loc_on`. When this transition is taken, the on-signal is sent via the synchronization label `sl_b1_sAon`.

The same kind of nondeterminism is used to determinate the stay in location `loc_on`. The invariant of the location is used to express the possibility how long we may stay

```
automaton b1_controller
    synclabs:
        sl_start_transfer_bA_b1, sl_stop_transfer_bA_b1,
        sl_b1_go, sl_b1_stop,   sl_b1_sAon, sl_b1_sBon,
        sl_start_transfer_b1_bB, sl_stop_transfer_b1_bB;
    initially loc_start;
    loc loc_start:
        while True wait {}
        when True sync sl_start_transfer_bA_b1 goto loc_start_1;
    loc loc_start_1:
        while True wait {}
        when asap sync sl_b1_go goto loc_transfer_bA_b1;
...
    loc loc_transfer_b1_bB_1:
        while True wait {}
        when asap sync sl_b1_stop goto loc_start;
end
```

Fig. 6. The automaton modeling the controller

in this the location, and the guard of the transition to `loc_off_after` expresses the possibility that the sensor switches off earlier.

The other sensor is modeled with a very similar automaton. The only differences are the position of the sensor on the belt and the signals it sends out. In its present form, HyTech does not allow the definition and instantiation of parameterized modules. Because of this, we have to repeat the definition of the sensors with the changes for the second sensor.

Experiences: The nondeterminism offered by HyTech could be used to model physical aspects of the system which are not really of a nondeterministic nature, but here we wanted to abstract from the detailed physical properties.

The component for sensors had to be replicated twice almost identically. The possibility to define parameterized automaton types and instantiating them would have been helpful make this identical structure explicit.

4.6 Modeling the controller

The automaton partly displayed in Figure 6 models the control program of the belt. We already described the algorithm in Section 2.2. Here, the algorithm is implemented as a HyTech component automaton.

The control program uses a large number of synchronization labels. Four of them are used for communication with the control programs of the neighbours:

- The synchronization label `sl_start_transfer_bA_b1` is used to start a transfer from the neighbour at endpoint `A` to the belt.
- After the belt has determined that the plate has been received, the signal `sl_stop_transfer_bA_b1` is sent to the neighbour.

– The signals `sl_start_transfer_b1_bB` and `sl_stop_transfer_b1_bB` are used analoguously, but here, the belt is the sending component and the neighbour at `B` is the receiving component.

The four other signals are used for communication with the belt hardware:

– `sl_b1_go` and `sl_b1_stop` are signals for the actuators of the belt. They control the belt movement.
– `sl_b1_sAon` and `sl_b1_sBon` are signals from the two sensors of the belt. They tell the control program that a plate has reached the given sensor.

As can be seen, global variables are not referenced by the control program. It only reacts to discrete signals, which are modeled with the synchronization labels. In some sense, these synchronization labels have to be translated by the automaton: The signal `sl_start_transfer_bA_b1` with which the neighbour at `B` starts the transfer must be translated to the signal `sl_b1_go` which sets in belt in motion. The location `loc_start` and `loc_start_1` accomplish this together. The keyword `asap` in location `loc_start_1` marks this transition as urgent. After time has passed, this transition cannot be taken any longer. For the automaton to continue, the signals `sl_start_transfer_bA_b1` and `sl_b1_go` must occur in the same time instant.

This kind of translation of an input signal into an output signal is typical for this control automaton. It is reflected in the pairwise occurrence of the locations: The first element of each location pair receives a signal, the second element emits a corresponding signal, but both events happen at the same instant.

HyTech only allows one synchronization label to be associated with every transition. If this restriction was lifted, and if a transition with several synchronization labels would have to synchronize with all these labels, the helper locations could have been avoided.

Experiences: The control program of the belt could be modeled as a HyTech component automaton. As a control program, this automaton is supposed to be deterministic. It would have been nice to express this explicitly and to have it checked by the system automatically.

Another point is that we missed the possibility to synchronize with several synchronization labels on one transition. This problem has been solved by using urgent helper locations which are offered by HyTech, but this technique generates more locations than would be necessary in our case, and it also makes it necessary to detect urgent configurations in some verifications. This last problem is talked about in Section 4.8.

4.7 Modeling the environment

The automaton in Figure 7 models the environment of the belt. Six synchronization labels are used for this. Four of them are used for the communication of the control programs of the neighbouring components. They have been described in Section 4.6 where the automaton describing the control of the belt was described. The other two,

```
automaton context
    synclabs:
        sl_bA_to_b1, sl_b1_to_bB,
        sl_start_transfer_bA_b1, sl_stop_transfer_bA_b1,
        sl_start_transfer_b1_bB, sl_stop_transfer_b1_bB;
    initially loc_start;
    loc loc_start:
        while True wait {}
        when True sync sl_start_transfer_bA_b1 goto loc_transfer_bA_b1;
    loc loc_transfer_bA_b1:
        while True wait {}
        when True sync sl_bA_to_b1 goto loc_transfer_bA_b1_1;
    loc loc_transfer_bA_b1_1:
        while True wait {}
        when asap sync sl_stop_transfer_bA_b1 goto loc_wait_for_plate;
...
    loc loc_receive_rest:
        while True wait {}
        when True sync sl_stop_transfer_b1_bB goto loc_start;
end
```

Fig. 7. Automaton modeling the environment of the belt

`sl_bA_to_b1` and `sl_b1_to_bB`, are used by the physical model describing the movement of the plate from one component to the other.

Thus, the environment automaton models both the control component of the environment and its physical aspects.

The environment modeled here is doing the following:

- After some time has passed, the control program of the neighbour at `A` tries to initiate the transfer of a plate (transition from `loc_start` to `loc_transfer_bA_b1`).
- Possibly after some time, the hardware of neighbour `A` signals that the plate is undergoing transfer (transition from `loc_transfer_bA_b1` to `loc_transfer_bA_b1_1`).
- Possibly after some more time, neighbour `A` receives the message that the transfer has been accomplished (transition from `loc_transfer_bA_b1_1` to `loc_wait_for_plate`).
- After some time, the control program of neighbour `B` receives the message that the belt wants to transfer a plate (transition from `loc_wait_for_plate` to `loc_receive_from_b1`).
- After some time, the hardware of neighbour `B` receives the plate (transition from `loc_receive_from_b1` to `loc_receive_rest`).
- Finally, after some more time the neighbor `B` sends the belt the message that the plate has been transferred. The cycle starts from the beginning (transfer from `loc_receive_rest` to `loc_start`).

As can be seen, a lot of possible environment behaviours are excluded by this automaton, but this is not a fundamental problem. Automata where new plates might be offered

```
automaton test1_bA_to_b1
    synclabs:
        sl_bA_to_b1, sl_stop_transfer_bA_b1;
    initially loc_start;
    loc loc_start:
        while True wait {dtest1_bA_b1_crit_time = 0}
        when asap goto loc_uncrit;
        when asap goto loc_crit;
    loc loc_error:
        while True wait {dtest1_bA_b1_crit_time = 0}
        when True sync sl_bA_to_b1 goto loc_error;
        when True sync sl_stop_transfer_bA_b1 goto loc_error;
    loc loc_uncrit:
        while True wait {dtest1_bA_b1_crit_time = 0}
        when True sync sl_bA_to_b1 goto loc_crit;
        when True sync sl_stop_transfer_bA_b1 goto loc_error;
    loc loc_crit:
        while True wait {dtest1_bA_b1_crit_time = 1}
        when True sync sl_bA_to_b1
            do {test1_bA_b1_crit_time' = 0} goto loc_error;
        when True sync sl_stop_transfer_bA_b1
            do {test1_bA_b1_crit_time' = 0} goto loc_uncrit;
end
```

Fig. 8. Component of the test automaton checking for a critical time interval

by neighbour `A` before the old plate has been received by neighbour `B` can be modeled easily. The control program should ensure that there are never two plates at the same time on the belt, and this might be checked with a test automaton.

4.8 A test automaton

At the end of Section 4.3, we mentioned a test automaton with which we wanted to check some assumptions about the cooperation of some inexactly modeled aspects of the physical model and the control program. We decided to model the transition of a plate from one component to another not physically correct for all possible cases, but only for the case that the control program worked as expected: During the transition of a plate from one component to another, both components are moving.

We describe this test automaton as the parallel composition of three component automata. Two of them check if the system is in a critical situation where the belt should be moving. One of them does this by recognizing the time interval of a transfer from neighbour `A` to the belt, and the other does this for the time interval of a transfer from the belt to neighbour `B`. The first of these two is given in Figure 8, the second has the same structure and just uses a different set of signals and another variable. The third component automaton checks if the belt is in motion or fixed. This one is given in Figure 9.

The forbidden situation is one in which the belt is stopped, i.e. `test1_going_stopped` is in location `loc_stopped`, and one of the

```
automaton test1_going_stopped
    synclabs:
        sl_b1_go, sl_b1_stop;
    initially loc_start;
    loc loc_start:
        while True wait {dtest1_stopped_time = 0}
        when asap goto loc_stopped;
        when asap goto loc_going;
    loc loc_error:
        while True wait {dtest1_stopped_time = 0}
        when True sync sl_b1_go goto loc_error;
        when True sync sl_b1_stop goto loc_error;
    loc loc_stopped:
        while True wait {dtest1_stopped_time = 1}
        when True sync sl_b1_go
            do {test1_stopped_time' = 0} goto loc_going;
        when True sync sl_b1_stop
            do {test1_stopped_time' = 0} goto loc_error;
    loc loc_going:
        while True wait {dtest1_stopped_time = 0}
        when True sync sl_b1_go goto loc_error;
        when True sync sl_b1_stop goto loc_stopped;
end
```

Fig. 9. Components of the test automaton checking for moving or stopped belt

other two test automata is in location `loc_crit`. This situation is not totally forbidden, but it is only allowed when no time passes while the system is in this configuration. To check if there exists such a situation, we have to be able to measure the time which has passed. For this, we need three further analogue variables which are used to measure the time the system is in the critical locations. For the displayed automata, these are the variables `test1_bA_b1_crit_time` and `test1_stopped_time`.

The components of the testing automaton should not restrict the rest of the system in any way, since it should only record what happens in the analysed system and not change the behaviour of this system. We will call an automaton which has no impact on the rest of the system an **input automaton**. All signals used in an input automaton should be input signals, and global variables controlled in an input automaton should not be used anywhere in the system.

In our case, we use a special structure for input automata. There are two special locations and a set of ordinary locations. The special locations are called `loc_start` and `loc_error`. `loc_start` is the initial location of the automaton. There are urgent transitions from this location to all ordinary locations. This construction is used to simulate several initial locations. `loc_error` is the error location. All transitions leading into this location lead back to itself.

The signals of these automata are implemented as true input signals, that is, the invariant in all the states is just `True`, as is the guard of each labeled transition, and from every location, there is a transition to another location for each synchronization label of the automaton.

`loc_error` is an error location. Transitions which should never be taken lead to this location. This makes possible a first check of the system: We might check if, from a given configuration, another configuration can be reached where one of the test automata is in location `loc_error`.

The more complicated correctness condition has been described before: We have to prove that in a critical time interval, the belt is always moving. The naive way would be to check, for our test automata, if a configuration can be reached where `test1_bA_to_b1` is in location `loc_crit` and at the same time, `test1_going_stopped` is in the location `loc_stopped`.

This naive way does not work. Our controller program automaton uses an urgent transition between the receipt of the signal that a plate is going to be transfered and the starting of the belt. Before this transition has been taken, a situation might be reached which `test1_bA_to_b1` classifies as critical while the belt has not been started. Our naive approach would detect that this situation is reachable and thus signal an error in the system.

But the transition of the controller is urgent: no time can pass between the receipt of the signal that a plate should be transfered and the starting of the belt. Thus, at the same instant in which the critical time interval might start, the belt is set in motion.

The critical situation we described would only indicate an error if time was allowed to pass in it. We use the three analogue variables starting with `test1_` to measure the time we stayed in a critical location for each automaton. The time derivatives and the reset assignments are set up with this goal. Now, the truly critical situations can be formalized by the already described condition for the locations and the further condition that both analogue variables are strictly positive. We will show an example of this in Section 5 on verification.

Experiences: It was nice to be able to model the test automaton with the same formalism as the system tested, but we would have wished to express the fact that these automata are purely input automata. This is a fact which might be checked automatically, or there might be a special syntax for this type of automata which excludes that the automaton restricts other automata in the system description.

It would also have been nice to allow several inital locations, but we were able to simulate this with HyTech's urgent transitions.

We had to introduce special analogue variables which measured the time spent in critical locations. This problem might be solved by an extension of the analysis language: If there was a predicate determining, for a given region, if time can pass while the system is in this configuration, we could have avoided the extra variables.

5 Verification

5.1 A plausibility check

Verification does not mean in this field that the truth of a model is proved in a formal way. The modeled reality is not in the reach of formal descriptions, and we use abstraction techniques to further simplify even the theoretically possible formal descriptions.

```
var reg_start, reg_reachable, reg_interest, reg_test_error: region;

-- The starting region.
reg_start :=
      b1_pos >= b1_length+plate_radius
    & loc[b1_movement] = loc_stopped_and_unloaded
    & loc[b1_sensA] = loc_off_after
    & loc[b1_sensB] = loc_off_after
    & loc[b1_controller] = loc_start
    & loc[context] = loc_start
    & test1_stopped_time = 0
    & test1_bA_b1_crit_time = 0
    & test1_b1_bB_crit_time = 0
    & loc[test1_bA_to_b1] = loc_uncrit
    & loc[test1_b1_to_bB] = loc_uncrit
    & loc[test1_going_stopped] = loc_stopped
    ;

-- Define interesting region: b1_pos is behind sensorB
--   and before the end of the belt.
reg_interest :=
  (b1_pos > b1_sensBpos+plate_radius) & (b1_pos < b1_length-plate_radius);

-- Print start region.
prints "Start region:";
print reg_start;
prints "";

-- Print region of interest.
prints "Region of interest:";
print reg_interest;
prints "";

-- Compute from reg_start reachable region.
reg_reachable := reach forward from reg_start endreach;

-- Print reachable region.
prints "Reachable region:";
print reg_reachable;
prints "";

-- Print trace from start to region of interest.
if empty( reg_interest & reg_reachable )
then
    prints "No way to region of interest";
else
    prints "Trace to region of interest:";
    print trace to reg_interest using reg_reachable;
endif;
```

Fig. 10. Analysis commands to check reachability of a region which should be reachable

We use the term here for the application of plausibility and consistency checks. We

give an example for both types by applying the HyTech analysis tools to the modeled system.

In the first check, displayed in Figure 10, we try to prove that, from a given configuration, a given region of interest can be reached.

In the second check, displayed in Figure 11, we try to prove that we cannot reach an erroneous situation characterized by the test automata.

Figure 10 contains the start of the analysis commands. Four variables are declared which will contain different regions: One is used for the starting region, one will contain the region reachable from the starting region, one will be used for the region of interest for which we want to have a series of transitions, and one will be used for the set of configurations which indicate an error via the test automaton.

We explain some notation with the example of the assignment of the starting region in which we are interested in to the variable `reg_start`. In this case, we use a conjunction of simple expressions. Each simple expression is either a linear comparison for values of the global variables, or a restriction of the location a component automaton is in. In the example, we describe a configuration where there is no plate on the belt, the belt is stopped, and the automata for the controller and the environment are in there start locations. The test automata components are set accordingly.

The region of interest is defined afterwards. The next six statements print newline-delimited commentary strings and newlines and the two regions just contructed in variables.

The right hand side of the assignment to `reg_reachable` computes a region representing all configurations reachable by forward transitions in the given automaton from the defined starting region. Afterwards, this reachable region is printed.

The `if`-expression terminating the example first checks if the reachable region intersects the region of interest. If this intersection is empty, a message is printed. In the other case, we print a trace, i.e. a sequence of time transitions and discrete transitions from the starting region to the region of interest. For the system described, the intersection is nonempty.

Experiences: The analysis language is easy to use for a programmer used to ordinary computing languages. It would have been nice to have some checks generated automatically, e.g. the avoidance of error states.

A temporal logic whose formulas are compiled into hybrid automata might a be more consise notation for formulating properties of an automaton than a test automaton.

5.2 The test automata

The check given in Figure 11 checks reachability of a forbidden region of the test automaton in a way totally analoguously to the first check. The expression which is assigned to the region of interest is a disjunction of simple expressions or disjunctions. The first three disjuncts describe the error locations of the components of the test automaton, the last two disjuncts describe the configurations for which our physical model is incorrect. The logic behind the checks is discussed in Section 4.8.

```
-- Use the testing automaton to check for erroneous configurations.
reg_test_error :=
        -- Wrong sequence of signals for transfer from bA to b1.
      loc[test1_bA_to_b1] = loc_error
        -- Wrong sequence of signals for transfer from b1 to bB.
    | loc[test1_b1_to_bB] = loc_error
        -- Wrong sequence of start/stop-signals.
    | loc[test1_going_stopped] = loc_error
        -- Belt stopped in critical time interval during
        --  transfer from bA to b1.
    | (  loc[test1_bA_to_b1] = loc_crit
       & loc[test1_going_stopped] = loc_stopped
       & test1_bA_b1_crit_time > 0 & test1_stopped_time > 0
      )
        -- Belt stopped in critical time interval during
        --  transfer from b1 to bB.
    | (  loc[test1_b1_to_bB] = loc_crit
       & loc[test1_going_stopped] = loc_stopped
       & test1_b1_bB_crit_time > 0 & test1_stopped_time > 0
      )
    ;

-- Check reachability of error region and trace.
if empty( reg_test_error & reg_reachable )
then
    prints "No way to error region of testing automaton.";
else
    prints "Trace to error region of testing automaton:";
    print trace to reg_test_error using reg_reachable;
endif;
```

Fig. 11. Analysis commands to check reachability of a forbidden region

Experiences: For component automata with explicit error locations, automatic construction of error regions and corresponding rechability tests would be convenient.

6 Summary

The theoretical base was sufficient in this case study. Analogue physical quantities could be modeled as having a piecewise constant derivative. A lot of the more advanced possibilities of HyTech have not been used in this case study. It must only be said that our model is too small to ensure that whole production cells can be analysed successfully.

The propositions we have regard enhancements of the notation:

- **Error location.** A special name for an error location together with automatically generated reachability tests would automate some checks.
- **Input and output signals.** Synchronization via synchronization labels worked nicely. But it would have been convenient to declare labels as either representing input or output signals of an automaton. For input signals, it could then be checked

that they are really receivable in every configuration of a component automaton, or incomplete transition sets could be completed with transitions into the error location.

- **Parameterized automaton types or templates.** Structuring the specification with parallely executed component automata was very helpful, but reuse of a component automaton once designed could be made easier by basing the notation on automaton types instead of on automaton instances. These types would be instantiated with some parameters for externally visible synchronization labels, analogue variables and constants. Thus, repeating similar elements, like the synchronization labels in our case, could be specified easier by reusing the designed automaton type, and hand checks of the models could also be easier.
- **Multiple synchronization labels in transitions.** In some situations, especially when we specified the controller, we could have used the possibility to label a transition with several synchronization labels. This would have meant that the transition would have to synchronize on all of its synchronization labels. In this way, we could also have avoided the urgent helper-states in the controller and the global variables measuring time in the critical locations of the testing automaton.
- **Time-extension-predicate for regions.** In the test automata, we had to introduce analogue variables just to check that the system could not stay in an illegal state for any time. It would have been convenient to have a predicate that checks if the system configuration can stay in the region while some time passes.
- **Hierarchy.** Some component automata are more closely associated than others. The notation could make this hierachical structure explicit by grouping associated component automata together in a medium level composite automaton. In our case study, we could have used this for grouping together the component automata for the physical properties and the component automata of the test automaton.
- **Controlled variables of an automaton.** Declaring an analogue variable as controlled by a specific automaton could make it explicit and checkable that the values of such a variable are not restricted outside a component automaton. Derivatives for the variable may only be restricted in this automaton, and discrete value transitions for the variable may also be given only in this automaton. This concept may be used together with hierarchy if different component automata are responsible for determining the derivatives and the discrete value changes of a variable.
  Automaton types could make this less necessary.
- **Input automata.** Declaring a component automaton as a pure input automaton. This property could either be checked or enforced by a special syntax for this kind of automata. Thus, there would be no danger that an automaton meant only for testing restricts the behaviour of the system.
- **Deterministic automata.** A component automaton could be declared as deterministic, and this property could be checked automatically. In this way, it could be checked that a component automaton is implementable as a control program for the system, or the control program could even be generated automatically.
- **Temporal logic.** A real time temporal logic might be used for the concise specification of properties of the system to be proved.

We consider the theoretical basis of HyTech as being strong enough for this kind of application, but for practical applications, the notation must be enhanced.

7 Acknowledgements

This work was inspired by the HyTech. Therefore, I want to thank HyTech's authors for making their tool available. Discussions in our group led to detection of shortcomings in previous versions of the paper. With Dirk Beyer, I discussed content, and with Claus Lewerentz, I discussed form. I wish to thank them to help me expressing myself better.

References

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[BBC+96] Nokolaj Bjørner, Anca Browne, Eddie Chang, Michael Colón, Arjun Kapur, Zohar Manna, Henny B. Sipma, and Tomás E. Uribe. STeP: Deductive-algorithmic verification of reactive and real-time systems. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the 8th International Conference on Computer Aided Verification (CAV'96)*, LNCS 1102, pages 415–418, Berlin, 1996. Springer-Verlag.

[BLL+96] Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Petersson, and Wang Yi. Uppaal – a tool suite for automatic verification of real-time systems. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Hybrid Systems III*, LNCS 1066, pages 232–243, Berlin, 1996. Springer-Verlag.

[DOTY96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Hybrid Systems III*, LNCS 1066, pages 208–219, Berlin, 1996. Springer-Verlag.

[HHWT95] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. A user guide to HyTech. In *Proceedings of the First Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS 1019, pages 41–71. Springer-Verlag, 1995.

[LL95] Claus Lewerentz and Thomas Lindner. *Formal Development of Reactive Systems*. Springer-Verlag, Berlin,Heidelberg, 1995.

[ORSvH95] Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995.