

# Specification and Verification of Distributed Technical Systems with Central Control

Gerhard Schellhorn\*  
Institut für Logik, Komplexität und Deduktionssysteme  
Universität Karlsruhe  
76128 Karlsruhe, Germany  
email: schellhorn@ira.uka.de

January 1994

## Abstract

This paper presents an algebraic approach to the specification and verification of distributed technical systems, which are controlled by a central control program. The approach is demonstrated by its application to the case study “production cell”. The approach uses first-order specifications to describe the possible behaviour of the system. Specifications are structured according to the physical structure of the system. A PASCAL-like program is used to enforce intended behaviour. The whole case study, including specification as well as verification of liveness and safety conditions, is carried out using the KIV system.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>The Intended Model</b>	<b>2</b>
<b>3</b>	<b>Algebraic Specification of State Oriented Systems</b>	<b>2</b>
3.1	System Independent Specification of Devices . . . . .	4
3.2	Specification of Devices in the Production Cell Context . . . . .	6
3.3	The System Specification . . . . .	7
<b>4</b>	<b>The Control Program and its Correctness</b>	<b>9</b>
<b>5</b>	<b>Verification</b>	<b>11</b>
<b>6</b>	<b>Conclusions and Further Work</b>	<b>13</b>
<b>A</b>	<b>The Specifications</b>	<b>14</b>
A.1	Specification of Events . . . . .	14
A.2	System Independent Specification of Devices . . . . .	16
A.3	Specification of Devices in the Production Cell Context . . . . .	33
A.4	The System Specification . . . . .	35
<b>B</b>	<b>The Control Procedure</b>	<b>40</b>
<b>C</b>	<b>The Invariant</b>	<b>44</b>

---

\*This research was sponsored by the BMFT project KORSO.

# 1 Introduction

The case study “production cell”, as described in [Li 93], [LL 94], was treated by the KIV group to study how distributed systems can be modelled within first-order logic, and which requirements for the correctness of a central control program can be expressed and verified using the KIV system. We did a complete formal development, including specification, implementation of a control program and verification.

The approach uses structured algebraic specifications to model possible behaviour of the devices of the system. A separately developed control program over the specification, implemented in a PASCAL-like notation is used to enforce intended behaviour. Verification of liveness and some aspects of safety has been done using the KIV system ([HRS 89], [HRS 90], [Re 92]), a system designed for the development of correct software systems. The system supports structured first-order specifications and a tactical theorem proving approach for program verification based on Dynamic Logic ([Ha 79]).

The following section describes the intended state oriented model for the system. Section 3 gives an overview of the specification. Section 4 describes the control program and gives the relevant correctness problems. Section 5 discusses the problem of verifying safety and liveness conditions and section 6 concludes.

## 2 The Intended Model

The choice of the intended model was mainly motivated by the attempt to mimic a (single) sequential control program that can be used to drive the production cell in reality. Such a program is usually called when some sensor value has changed (significantly), like “press has reached upper position”. Activation of the control program is done either by an interrupt to the controlling computer or by a polling routine. The program reacts on the change by giving a number of controlling commands to the actuators, like “move press downwards”. Formally speaking the program is a routine, which gets a “sensor event” as input, and reacts on it, by giving as output some “control events” to the system. To make the program react properly, it must keep an abstract state in a global variable, which reflects the relevant properties of the physical devices, such as “number of metal blanks on feed belt” etc.. So the control program abstracts from reality to the computer representable state of an (not necessarily finite) automaton, where both sensor and control events correspond to state changes. Time is treated implicitly by assuming calls to the control program, whenever sensor data signal a state change in the system. Using a control program which abstracts reality to a state oriented representation, we have chosen to model the devices by a specification that uses states and events too.

It should be noted that such a state oriented control program works only under the assumption, that the reaction of the computer is fast enough to avoid significant changes in the devices while it is running. As an example, if a robot arm signals that it has reached an angle, where it is in front of the press, the program should react fast enough to stop the arm at an angle, where it is still in front of the press. Since this assumption, which can roughly be formulated as “execution time does not matter” is widely used and did not seem critical in the concrete scenario, we adopted it. It should nevertheless be noted, that the assumption is essential for the abstract model described in the following. Dealing with execution times would have required a much more complicated specification dealing with explicit time or interrupts with priorities.

## 3 Algebraic Specification of State Oriented Systems

Modelling a system by a formal algebraic specification always requires choosing a suitable abstraction from the real world and should give a specification whose structure should reflect the one of the real system.

In the case of the production cell the specification structure is naturally given by the devices of the system, so we chose to specify each component separately.

For the abstraction level, three choices are possible: The first is to model reality as close as possible, by specifying the state of a device as a tuple of all available sensor values. This would require to specify the robot as a triple of potentiometer values, and every change in those values would change the state of the device by an event “value increased” or “value decreased”. The second choice is the abstract state, which the control program uses to compute its answers. Since these data types must be specified anyhow, this is the easier choice. We actually used a third, even more abstract level by simplifying the description of the moving devices robot, elevating rotary table and travelling crane. We assume that we can send the signal “move to feed belt” to the travelling crane and get the response “arrived at feed belt” when it reaches this position, stopping automatically. An approach closer to reality would have been to add a control event “stop at current position”, but this would have simply increased the number of states and events, without adding anything essentially new to the problem. A more interesting point is that all three levels are linked by abstraction functions, and it would be an interesting task to study the relation between them. Several possibilities for the realization of the translation between two levels are of interest:

1. Intelligent control at the devices
2. Parallel processes for every device on the computer on which the control program is executed
3. A preprocessing control program that is placed on top of the current control program
4. Direct translation of the control program to a more detailed one

The first two possibilities are beyond the concept of a sequential control program and therefore not in the scope of the approach used here, while the latter two seem to be capable by defining a suitable refinement relation between specifications and programs.

In the following we will describe the specification, fully listed in appendix A, in detail. The specification consists of three parts, described in the following sections: On the bottom level, we have two specifications for every device, describing the events and states of the individual devices. These specifications are independent of their actual use in the production cell and are subject to reuse. At a second level we have a specification for every device that describes the restrictions imposed by the use of the device in the context of the production cell, such as where it is loaded or where it is allowed to move. This specification may vary in different contexts. On the top level the system specification of the production cell is defined by composition of the devices specification and defining the interaction between them.

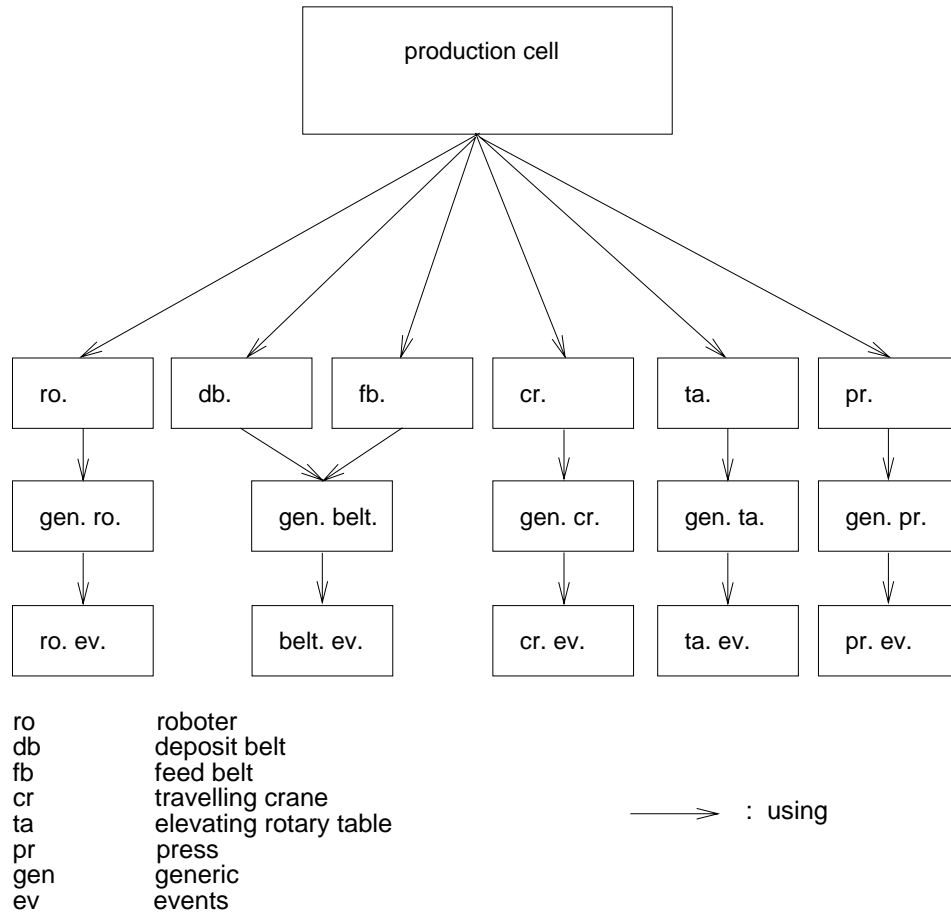


Fig.1: global specification

### 3.1 System Independent Specification of Devices

Since we adopted a state oriented model the specification of every device consists of two parts: a specification of the possible events and a specification of the possible states of the device. Events are classified as sensor events (events from the physical system, i.e. significant sensor value changes) and control events (actions taken by the control program, driving the actuators). For the elevating rotary table we get the specification:

```

TABLE_EVENTS =
data specification
  ta_vposition = ta_up | ta_down;
  ta_hposition = ta_fb | ta_ro;
  ta_event     = (* sensor events *)
                ta_hat(ta_hpos : ta_hposition) | ta_vat(ta_vpos : ta_vposition) |
                ta_load | ta_unload |

                (* control events *)
                ta_hto(ta_hpos : ta_hposition) | ta_vto(ta_vpos : ta_vposition);

end data specification
  
```

The specification describes a free data type of ten different events represented by ground terms. Events are divided into the sensor events “table arrives at upper position” ( $ta\_vat(ta\_up)$ ), “table arrives at lower position” ( $ta\_vat(ta\_down)$ ), “table arrives at feed belt” ( $ta\_hat(ta\_fb)$ ), “table arrives at robot” ( $ta\_hat(ta\_ro)$ ), and the control events “load table” ( $ta\_load$ ), “unloaded table” ( $ta\_unload$ ), “move to upper position” ( $ta\_vto(ta\_down)$ ), “move to lower position” ( $ta\_vto(ta\_up)$ ), “rotate to feed belt” ( $ta\_hto(ta\_fb)$ ) and “rotate to robot” ( $ta\_hto(ta\_ro)$ ). It uses two boolean data types for the horizontal (i.e. the rotation angle) and the vertical position, which can be selected from the events with the selectors  $ta\_hpos$  and  $ta\_vpos$ . The positions “at feed belt” ( $ta\_fb$ ) and “at robot” ( $ta\_ro$ ) are independent of the concrete use of the device, but to guide intuition we have indicated their further use by appropriate identifiers. The specification of possible states of the elevating rotary table looks as follows:

GENERIC\_TABLE = **enrich** TABLE\_EVENTS **by**

**sorts**         $ta\_state$ ;

**constants**    $taerr : ta\_state$ ;

**functions**     $ta\_hpos : ta\_state \rightarrow ta\_hposition$ ;  
                   $ta\_vpos : ta\_state \rightarrow ta\_vposition$ ;  
                   $ta\_trans : ta\_event, ta\_state \rightarrow ta\_state$ ;

**predicates**    $ta\_hmoves, ta\_vmoves, ta\_loaded : ta\_state$ ;  
                   $ta\_exp : ta\_event, ta\_state$ ;

**variables**     $tas, tas1, tas2 : ta\_state$ ;

**axioms**

$tas1 \neq taerr$   
 $\rightarrow ( \quad tas1 = tas2$   
 $\quad \leftrightarrow \quad tas2 \neq taerr$   
 $\quad \wedge ta\_hpos(tas1) = ta\_hpos(tas2) \wedge ta\_vpos(tas1) = ta\_vpos(tas2)$   
 $\quad \wedge (ta\_hmoves(tas1) \leftrightarrow ta\_hmoves(tas2))$   
 $\quad \wedge (ta\_vmoves(tas1) \leftrightarrow ta\_vmoves(tas2))$   
 $\quad \wedge (ta\_loaded(tas1) \leftrightarrow ta\_loaded(tas2)) )$ ,

$ta\_vpos(tas) \neq ta\_up \wedge ta\_hmoves(tas) \rightarrow tas = taerr$ ,

$ta\_exp(ta\_hat(ta\_fb),tas) \leftrightarrow tas \neq taerr \wedge ta\_hmoves(tas) \wedge ta\_hpos(tas) = ta\_fb$ ,

....

$\neg ta\_exp(ta\_load)$ ,

....

$ta\_trans(ta\_hto(ta\_fb),tas) \neq taerr$   
 $\leftrightarrow \quad tas \neq taerr \wedge ta\_hpos(tas) = ta\_ro \wedge ta\_vpos(tas) = ta\_up$   
 $\quad \wedge \neg ta\_hmoves(tas) \wedge \neg ta\_vmoves(tas)$ ,

$tas1 = ta\_trans(ta\_hto(ta\_fb),tas) \wedge tas1 \neq taerr$   
 $\rightarrow \quad ta\_hpos(tas1) = ta\_fb \wedge ta\_vpos(tas1) = ta\_up \wedge ta\_hmoves(tas1)$   
 $\quad \wedge \neg ta\_vmoves(tas1) \wedge (ta\_loaded(tas1) \leftrightarrow ta\_loaded(tas))$ ,

$ta\_trans(ta\_load,tas) \neq taerr$   
 $\leftrightarrow \quad tas \neq taerr \wedge \neg ta\_hmoves(tas) \wedge \neg ta\_vmoves(tas) \wedge \neg ta\_loaded(tas)$

```

    tas1 = ta_trans(ta_load,tas) ∧ tas1 ≠ taerr
→   ta_hpos(tas1) = ta_hpos(tas) ∧ ta_vpos(tas1) = ta_vpos(tas)
    ∧ ¬ ta_hmoves(tas) ∧ ¬ ta_vmoves(tas) ∧ ta_loaded(tas),
....
end enrich

```

The specification describes the possible states of the elevating rotary table. On events, a new state is reached by applying the transition function `ta_trans` to the event and the old state. According to the first axiom, states are characterized by their horizontal and vertical position and their movement. If `ta_hmoves(tas)` is true, the table rotates (horizontally) to the position `ta_hpos(tas)`, if it is false, it is standing at that position. An additional state `taerr`, different from all other states, is used to model crashes of the table due to incorrect control. According to the task description (section 1.3.1, point 5) trying to rotate the table in all positions but the upper would result in such an error. This is reflected by the second axiom. The extension of the predicate `ta_exp` is the set of all sensor events which are expected to occur if the production cell is in state `tas`. They will play an important role as possible inputs for the control program. The third axiom, which represents a number of axioms for every event says that arrival at feed belt is expected if and only if the table moves towards that position. This predicate is also used to distinguish sensor and control events, since it is false in every state for control events. The rest of the axioms deal with the transition function. For every event a first axiom gives the safety conditions under which the event will not lead to the error state. A second axiom states the effects on the state, if this condition is fulfilled. For the event “rotate to feed belt” the transition will not lead to an error if and only if the table stands in upper position turned towards the robot (maybe this restriction is specific to the use of the table in the context of the production cell, so it should be moved to the following specification of context requirements, the task description does not indicate if this is the case). In the new state the table will rotate towards the angle where it is positioned towards the feed belt. Loading the table is acceptable in every position, in which the table does not move, so we have nothing assumed about the use of the device in the production cell.

The other devices are modelled in a similar way to the elevating rotary table. The specifications for the two belts coincide, they both have the same functionality. We assume a photoelectric barrier at the end of the feed belt too, which is a slight change to the task description. Some change is necessary at this point since we must be able to detect when a metal plate has moved on the rotary table. Another possibility would have been to install a timing mechanism on the computer, which would be started when starting the feed belt with a metal plate. After the time it takes the plate to arrive at the end of the belt it would signal an appropriate event. Introduction of timers or alternatively a second photo electric barrier at the front of the feed belt would have been more flexible and would have allowed to place more than one metal plate on the belts, which is not the case in the current system.

### 3.2 Specification of Devices in the Production Cell Context

Using the elevating rotary table in the context of the production cell yields restrictions on the positions, where the table should be loaded and unloaded. These restrictions are modelled by a new transition function `ta_tr`, which yields the error state if the device is loaded and not in lower position turned towards the feed belt. We get the following specification:

TABLE =

**enrich** GENERIC\_TABLE **by**

**functions** ta\_tr : ta\_event,ta\_state  $\rightarrow$  ta\_state;

**axioms**  $\neg$ (tae = ta\_load  $\vee$  tae = ta\_unload)  
 $\rightarrow$  ta\_tr(tae,tas) = ta\_trans(tae,tas),

$\neg$ (ta\_hpos(tas) = ta\_fb  $\wedge$  ta\_vpos(tas) = ta\_down)  
 $\rightarrow$  ta\_tr(ta\_load,tas) = taerr,

ta\_hpos(tas) = ta\_fb  $\wedge$  ta\_vpos(tas) = ta\_down  
 $\rightarrow$  ta\_tr(ta\_load,tas) = ta\_trans(ta\_load,tas),

$\neg$ (ta\_hpos(tas) = ta\_ro  $\wedge$  ta\_vpos(tas) = ta\_up)  
 $\rightarrow$  ta\_tr(ta\_unload,tas) = taerr,

ta\_hpos(tas) = ta\_ro  $\wedge$  ta\_vpos(tas) = ta\_up  
 $\rightarrow$  ta\_tr(ta\_unload,tas) = ta\_trans(ta\_unload,tas),

**end enrich**

For other devices the specification contains more restrictions than that for load and unload position, e.g. the restriction that the deposit belt may not drop a metal blank when the blank reaches the end of the belt is formulated here (a generic belt may do this, and in fact the feed belt does load the elevating rotary table by simply dropping the blank in the event “fb\_blankfalls”). All safety requirements that affect the use of devices in the context of the production cell are formulated in the specification of its use in the production cell context. The only exceptions are conditions 1 and 2 in section 1.3.1 of the task description, since they are concerned with the interaction of two devices (press and robot).

### 3.3 The System Specification

The system specification composes the specifications of the devices. The states of the whole system are the cartesian product (built by mkstate, decomposed by fbc,tac etc.) of the states of the individual devices, while the events of the system are the union of the device events. In this union corresponding load and unload events are identified. Conversion functions cfb, cta etc. are used to convert from the events of the individual devices to the events of the system. The transition function trans of the system calls the transition functions of the devices for their events. Safety is formalized in the system specification by a predicate that is true for a state, if no device is in error state and interaction between robot and press is appropriate (see conditions 1 and 2 of the safety requirements given in section 1.3.1 of the task description).

```

PRODUCTION_CELL =
  enrich

    data specification
      using FBELT, TABLE, ROBOT, PRESS, DBELT, CRANE
      state = mkstate(fbc : fb_state, tac: ta_state, roc : ro_state,
                     prc : pr_state, dbc : db_state, crc : cr_state)
    end data specification

  by

    sorts      event;

    functions  cfb : fb_event  $\rightarrow$  event;
               cta : ta_event  $\rightarrow$  event;
               cro : ro_event  $\rightarrow$  event;
               cpr : pr_event  $\rightarrow$  event;
               cdb : db_event  $\rightarrow$  event;
               ccr : cr_event  $\rightarrow$  event;
               trans : event, state  $\rightarrow$  state;

    predicates exp : event, state;
               safe : state;

    axioms
      (* conversion of events *)
      tae1 = tae2  $\leftrightarrow$  cta(tae1) = cta(tae2),
      fbe  $\neq$  fb_blankfalls  $\rightarrow$  cfb(tae1)  $\neq$  cta(tae2)),
      cfb(fb_blankfalls) = cta(ta_load),
      ....
      (* the transition function *)
      fbe  $\neq$  fb_blankfalls
       $\rightarrow$  trans(cfb(fbe),mkstate(fbs,tas,ros,prs,dbs,crs))
      = mkstate(fb_tr(fbe,fbs),tas,ros,prs,dbs,crs),

      trans(cfb(fb_blankfalls),mkstate(fbs,tas,ros,prs,dbs,crs))
      = mkstate(fb_tr(fb_blankfalls,fbs),(ta_tr(ta_load,tas),ros,prs,dbs,crs)),
      ....

      (* the expected events *)
      exp(cfb(fbe),mkstate(fbs,tas,ros,prs,dbs,crs))  $\leftrightarrow$  fb_exp(fbe,fbs),

      tae  $\neq$  ta_load
       $\rightarrow$  exp(cta(tae),mkstate(fbs,tas,ros,prs,dbs,crs))  $\leftrightarrow$  ta_exp(tae,tas),

```



```

safe(mkstate(fbs,tas,ros,prs,dbs,crs))
↔ fbs ≠ fberr ∧ tas ≠ taerr ∧ ros ≠ roerr
  ∧ prs ≠ prerr ∧ dbs ≠ dberr ∧ crs ≠ crerr
  (* do not crush arms with press *)
  ∧ ( ( ro_pos(ros) = ro_a1atpr ∧ ro_a1pos(ros) ≠ ro_ain
        ∨ ro_pos(ros) = ro_a1atpr ∧ ro_a1pos(ros) ≠ ro_ain)
      → ¬ pr_moves(prs) ∧ pr_pos(prs) ≠ pr_up)
  (* do not rotate with extracted arms *)
  ∧ ¬ (ro_moves(ros) ∧ (ro_a1pos(ros) ≠ ro_ain ∨ ro_a2pos(ros) ≠ ro_ain))

```

**end enrich**

The resulting system has the impressive total number of  $8 \times 32 \times 1152 \times 12 \times 8 \times 12 = 339.738.624$  states (error states of the devices and identification with them by axioms not considered) and  $4 + 7 + 17 + 7 + 4 + 11 = 50$  different events, 26 of them being sensor events.

The robot is clearly the most complex device with 4 relevant angles to rotate to, 3 positions per arm, 3 predicates indicating movement of the robot and its arms and 2 predicates indicating which arms are loaded. Although the specifications together have a length of 2000 lines (including comments) and contain 611 axioms, a third of which are generated automatically, they are easy to write (the only problem being copy and paste errors ...). It is easy to prove consistency of the specification, since most data types used are enumerations or record data types which are consistent by construction.

## 4 The Control Program and its Correctness

Having described the possible states, events and transitions of the production cell, we are now able to implement a procedure that controls the behaviour of the system. The procedure is a PASCAL program receiving a sensor event  $e$  as input (value parameter). It computes a list  $el = (e_1, \dots, e_n)$  of control events as output in its reference parameter. These are used to drive the actuators. The procedure uses a global variable  $v$  to model the current state of the physical system, that is updated according to the incoming sensor event and the computed control events. To prove correctness of the program, we have to show three characteristic properties:

1. the control program models reality adequately, i.e. the value of the variable  $v$  of the program reflects the physical state  $s$  of the system
2. the system is safe, it never reaches an erroneous state which does not satisfy the safety predicate
3. the system is life, i.e. it keeps on running forever.

Since we have chosen to model the physical system state and the abstract value of the program variable to be the same data type the “models” relation is simply expressed as equality (otherwise it would have been an abstraction function from physical states to values of the variable) and property 1 can be proved, showing that  $v = s$  is an invariant of the program. Written in Dynamic Logic the invariance assertion is

$$v = s \wedge \text{exp}(e, s) \rightarrow \langle \text{control}(e; el) \rangle v = \text{trans}^*(el, \text{trans}(e, s))$$

where  $\text{control}$  is the control procedure,  $\text{trans}$  is the transition function of the system, and  $\text{exp}$  is the predicate describing expected sensor events in state  $s$ . The assertion states formally, that if  $v$  describes the current state of the system, and  $e$  is an expected sensor event in  $s$  (i.e. a significant change in sensor values that may happen in the current state  $s$ ), then the control program terminates yielding a list of control events  $el$ , and the value of variable  $v$  will reflect the induced state change by the sensor event and the control actions computed. This state change

is computed formally by applying the transition function to first the event  $e$  and then to the list of events  $el$  (for  $el = (e_1, \dots, e_n)$ ,  $trans^*(el,s)$  is inductively defined to be  $trans(e_n, trans(\dots trans(e_1,s) \dots))$ ). Proving the correctness assertion is trivial in this case, where abstract system state and the model of the physical system coincide, since the program simply updates the variable  $v$  at the beginning with  $trans(e,v)$  and at the end with  $trans^*(el,v)$  using the same transition function as the physical system (if the models would differ the proof would become nontrivial).

To show how safety of the system can be guaranteed, we should first have a look at the program (160 loc, appendix B contains a full listing). Apart from these assignments described above, it is a case distinction on the 26 possible sensor events. In every case an answer is computed, sometimes with, sometimes without looking at the value of  $v$ . The following part of the control procedure gives two cases:

```

procedure control( $e$ ; var  $el$ )
begin
 $v := trans(e,v)$ ;  $el := nil$ ;
case  $e$  of
  ....
   $ta\_vat(ta\_up)$ :                                     (* table arrives in upper position *)
    begin  $el := cons(cta(ta\_hto(ta\_ro)),el)$  end          (* rotate to robot *)

   $cro(ro\_a2at(ro\_ain))$ :                                  (* arm2 has been retracted *)
    begin
      if  $ro\_pos(roc(v)) = ro\_a2pr$                        (* arm2 in front of press *)
      then begin
         $el := cons(cpr(pr\_to(pr\_mid)),el)$ ;             (* move press to loading position *)
        if  $ro\_alloaded(roc(v))$                           (* robot arm1 is loaded *)
        then  $el := cons(cpr(pr\_to(ro\_a1pr)),el)$          (* move with arm1 to press *)
        else if  $db\_loaded(dbc(v))$                       (* dep. belt is loaded *)
          then  $el := cons(cro(ro\_to(ro\_a1ta)),el)$        (* move with arm1 to e. r. table *)
          else  $el := cons(cro(ro\_to(ro\_a2db)),el)$        (* move with arm2 to deposit belt *)
        end
      else                                               (* robot is with arm2 at dep. belt *)
        if  $pr\_loaded(prc(v))$                             (* press is loaded *)
        then  $el := cons(cro(ro\_to(ro\_a2pr)),el)$          (* move with arm2 to press *)
        else if  $ro\_alloaded(roc(v))$                     (* arm1 is loaded *)
          then  $el := cons(cro(ro\_to(ro\_a1pr)),el)$        (* move with arm1 to press *)
          else  $el := cons(cro(ro\_to(ro\_a1ta)),el)$        (* move with arm1 to e. r. table *)
        end
      end
    end
  ....
end case;
 $v := trans^*(el,v)$ 
end

```

Depicted is the reaction on the event “robot arrives at upper position”, which is simply “rotate to robot” and the reaction on the event “robot arm2 has been retracted”. This is the most complicated case, since depending on where the robot is standing (in front of elevating rotary table or in front of deposit belt) control events must be computed, that decide where the robot has to be rotated to next and whether the press has to be moved. The program makes assumptions about the state it is in. E.g. it assumes that if arm2 has been retracted from the deposit belt, the arm has just been unloaded. These assumptions must hold, if the resulting state after the program should be safe. In the example, if arm2 were still loaded, and press were loaded too, arm2 would move to the press, and would try to pick up a second metal plate, resulting in an error. The assumptions can be intuitively formulated as “loaded devices move forward until they

reach the position to unload, and unloaded devices move backward until they reach the position to load”. Formalizing this safety invariant, we get a predicate  $\text{safe-inv}(s)$ , and we have to prove that

$$v = s \wedge \text{safe}(s) \wedge \text{safe-inv}(s) \wedge \text{exp}(e, s) \rightarrow \langle \text{control}(e; s) \rangle (\text{safe}(s) \wedge \text{safe-inv}(s))$$

holds. Then, started in an initial state, which satisfies  $\text{safe}(s) \wedge \text{safe-inv}(s)$ , our program will work in the sense that it will never reach an unsafe state.

To guarantee liveness of the system, we must assure that in any state reached by the system we still have an expected event  $e$ , for which  $\text{exp}(e, s)$  holds, which means that not all devices are stopped. This is not the case for all states satisfying  $\text{safe}(s)$  and  $\text{safe-inv}(s)$ , so we must impose further restrictions. Two obvious ones are that the production cell must contain metal plates, and that there must be a possibility to move at least one metal plate, which requires that one position is not loaded. Since the current description has seven positions where a metal plate may be (one for every device except the robot, who has two arms, that may be loaded), the number of plates the cell may handle, is restricted to 6. Another less obvious restriction is “loaded devices are that have reached their ‘unloading position’ are unloaded immediately if the device to load is ready”, for they would wait forever otherwise.

A very intricate problem here are the two robot arms, since their movement is not independent of each other. The two critical events are just the ones, when one robot arm has been retracted. One is shown in the program above. If we would follow the task description for the robot (section 1.1.3) literally, we would get a deadlock, if there were just one metal in the production cell, after picking up a piece from the elevating rotary table (step 1), since there is no blank in the press to get. Even if we assume a second piece in the press, we would get a deadlock after step 3, if all positions in the production cell except the arms of the robot are filled (then we now should first pick up a blank from the elevating rotary table). Formulating all these restrictions by a predicate  $\text{life-inv}(s)$ , we have to show that they are left invariant by the control program too, i.e.

$$\begin{aligned} v = s \wedge \text{safe}(s) \wedge \text{safe-inv}(s) \wedge \text{life-inv}(s) \wedge \text{exp}(e, s) \\ \rightarrow \langle \text{control}(e; s) \rangle (\text{safe}(s) \wedge \text{safe-inv}(s) \wedge \text{life-inv}(s)) \end{aligned}$$

and that these restrictions guarantee indeed liveness, i.e.

$$\text{safe}(s) \wedge \text{safe-inv}(s) \wedge \text{life-inv}(s) \rightarrow \exists e. \text{exp}(e, s)$$

## 5 Verification

Verification of the safety and liveness condition of the production cell is trivial from the theoretical point of view, since both conditions can be viewed as formulas of the propositional calculus (for the liveness condition the existential quantifier can be replaced by an explicit disjunction on the 50 possible events), which is decidable. So all that is required to prove the goals is an efficient prover for propositional logic. Our first impression of the case study was, that we could simply use the propositional rules of the sequent calculus built into the KIV-System together with appropriate rewrite rules. Then proofs could be done automatically.

Unfortunately, this attempt is not feasible, since proofs grow exponentially, when case distinctions (via the usual rules of sequent calculus for conjunctions in the succedent and disjunction in the antecedent) are applied without restrictions. The same proof trees simply get duplicated by doing case distinctions that are irrelevant to the current subgoal. As an example, if we want to prove a subgoal that states: “After some events initiated by the control program, press is loaded”, a case distinction on the initial position of the travelling crane will simply duplicate the subgoal. Since we wanted to do proofs with the KIV-System, we decided to give up the attempt to prove correctness automatically. To avoid doing all case distinctions by hand, we used the “module specific” heuristic to model the typical situations where a case distinction is appropriate (the name stems from the fact, that the heuristic was originally designed to handle typical situations

in proving module correctness). The “typical situations” used here are quite simple, they specify situations where at least one of the premises can immediately be seen to be an axiom. About 60 – 70% of the case distinctions can be done automatically using the heuristic, the rest must be given by hand.

To increase the readability of the sequents of the proof we used the “module specific” heuristic to weaken unnecessary preconditions from the proof too.

Apart from case distinctions and weakening steps the proof consists mostly of simplification steps, that reduce applications of the global transition function to applications of their local counterpart, eliminate applications of the local transition function by giving appropriate preconditions and deal with the inequality of constants (such as  $ta\_fb \neq ta\_ro$ ). A typical lemma used for simplification is:

$$\begin{aligned} & ta\_loaded(ta\_tr(ta\_vto(ta\_up),tas)) \\ \rightarrow & \quad ta\_tr(ta\_vto(ta\_up),tas) = ta\_err \\ & \vee ( \quad ta\_tr(ta\_vto(ta\_up),tas) \neq ta\_err \\ & \quad \wedge tas \neq ta\_err \wedge ta\_vpos(tas) = ta\_down \\ & \quad \wedge ta\_hpos(tas) = ta\_fb \wedge ta\_loaded(tas) \end{aligned}$$

The Lemma is used as a rule to rewrite every instance of the premise of the implication to the conclusion. Altogether we used over 1000 such rules, which demands a very efficient simplification strategy. All the lemmas used for simplification are axioms or propositional reformulations of axioms, so they can be proved easily. 200 of them are axioms from data specifications, which are used automatically.

A first version of the specification, the implementation of the control program, and an initial definition of the simplifier rules and the “typical situations” for the “module specific” heuristic can be derived in about two weeks of work.

But now we encounter the typical problems of verification: Trying to prove the liveness goal we directly run into an unprovable subgoal. Since the subgoal explicitly shows a state that is not life, i.e. one where no event is expected, the decision which part of our system is incorrect (specification, simplifier rules, program or one of the invariants *safe-inv* and *life-inv*) is easy, but as it turns out, the error we detected is not the only one.

Altogether we discovered about 30 errors during the verification process we have done so far. Most of the errors were discovered during the proof of the liveness goal. Some of the errors were purely syntactical, some concerned the strategy of the program but most of them resulted from missing properties in the invariants *safe-inv* and *life-inv*. One of the most intricate liveness properties is that the robot does not rotate with *arm1* to the elevating rotary table, if *arm1* is not loaded, *arm2* is loaded and *all* the other devices except *press* are not loaded (see the last conjunct in appendix C).

If we would have to start to prove liveness from scratch every time we discovered an error, we would have never reached any success. But fortunately the KIV-system can reuse the proofs of corrected goals, which saves a vast amount of time. Nevertheless proving the liveness is still quite a lot of work to do, and unfortunately we have not found a way to make the proof modular, i.e. we did not find a set of lemmata sufficient to prove the liveness goal, such that changes in the invariants would affect only *some* of the lemmata.

Starting with an initial version of the goal we arrived at a complete proof for the liveness assertion after about two weeks of work. The final version of the invariant

$$safe(s) \wedge safe\_inv(s) \wedge life\_inv(s)$$

can be found in appendix C. The statistic for the liveness proof is depicted below.

Safety still requires some more work. The initial goal first splits into 26 goals, one for every sensor event. Further conditionals in these cases (with a maximum of 6 for the event “robot *arm2* has been retracted”) give 51 cases to prove. The proof for every goal requires about a day

of work, so we did only 11 exemplary cases including the 6 cases of the event “robot arm2 has been retracted”. Proofs are so simple and tedious (and they all look very similar), that it is possible to deal with several proofs simultaneously (which was never possible in other case studies). Three typical statistics for the lifeness proof and the first two cases of the “robot arm2 has been retracted” look as follows:

	lifeness	roa2atin-case1	roa2atin-case2
proof steps	1316	731	951
simplification	603	432	525
weakening	295	100	174
case distinction	395	199	252
interactions	81	72	74

The diagram shows the number of proof steps required to proof the goal, which splits in simplification, weakening and case distinction steps (as described above). The interactive steps are all case distinctions.

## 6 Conclusions and Further Work

We have done a case study in specification and verification of a distributed technical system with a central control program. The algebraic approach was suitable to derive a structured specification of the system as a composition of reusable device specifications. We implemented a control program, that although it is not suitable to drive the production cell model of the FZI due to the chosen abstraction level, seems not too far away from a realistic application. Here the connection between different abstraction levels seems to be an interesting topic for further research. The tactical theorem proving approach used in the KIV-System was sufficient to prove the goals, although it seems that work has to be done in the modularization of correctness proofs to make them more feasible. For the finite state space used here, it should also be possible to use techniques of symbolic model checking, which would do proofs automatically and therefore seem to be more adequate. Maybe the invariants  $\text{safe-inv}$   $\text{life-inv}$  could be derived automatically too from the control program.

Tactical theorem proving may again become relevant, if we change towards a more realistic scenario adding suitable sensors or timers, to get rid of the “only one plate on each belt” restriction. Allowing any (finite) number of plates on a belt would turn the problem from a propositional logic one to a problem of predicate logic, where techniques operating on finite state spaces would be no longer applicable. Other steps towards a more realistic scenario include the introduction of a startup routine, or the possibility to add and subtract metal plates from the cell. A final point completely missing here is the comparison with other approaches to the specification of distributed systems. Connections to functional specifications using streams as data type should be clarified as well as the connections to specifications using temporal logic. Finally we wish to thank our students Markus Friedel and Farzad Safa for their work on this case study.

## A The Specifications

The specifications and control program listed below are the output of the conversion routine to  $\text{\LaTeX}$  of the `cosi` strategy. They differ in some minor points from the description above:

- Instead of a predicate `ta_hmoves` and two positions `ta_fb`, `ta_ro` four positions `ta_b1`, `ta_ro`, `ta_tobl`, `ta_toro` are used, the latter two indicating movement. The same for all predicates indicating movement
- The case-statement (not available in the PASCAL-dialect of the KIV system) of the control program is replaced by conditionals
- The list operations are named differently (see the specification `PROD_CELL_WITH_EVENT_LIST`)
- Instead of specifying one generic belt and renaming it, two isomorphic specifications are used
- For technical reasons it is necessary that the states of the devices have an explicit “generated by” clause. Therefore an (unspecified) initial state is introduced for every device.
- The names of variables are different

### A.1 Specification of Events

```
FBELT_EVENTS =
data specification
  fb_event = fb_load | fb_unload | fb_on | fb_off
             | fb_pbinterruption | fb_sheetfalls
             ;
  variables fbe1: fb_event;
end data specification

TABLE_EVENTS =
data specification
  ta_position = ta_up | ta_toup | ta_down | ta_todown ;
  ta_tposition = ta_fb | ta_tofb | ta_ro | ta_toro ;
  ta_event = ta_mvto (ta_mvtos : ta_position)
             | ta_tmvtto (ta_tmvttos : ta_tposition)
             | ta_load | ta_unload
             | ta_at (ta_ats : ta_position)
             | ta_tat (ta_tats : ta_tposition)
             ;
  variables taev1: ta_event; tapo1: ta_position; tatpo1: ta_tposition;
end data specification

ROBOT_EVENTS =
data specification
  ro_aposition = ro_ain | ro_atoin
                | ro_aout | ro_atoout
                | ro_amid | ro_atomid
                ;
  ro_position = ro_a1pr | ro_a1upr
                | ro_a2pr | ro_a2upr
                | ro_a1ta | ro_a1tota
                | ro_a2db | ro_a2todb
                ;
```

```

ro_event = ro_a1mv (ro_a1mvs : ro_ainposition)
           | ro_a1load | ro_a1unload
           | ro_a2mv (ro_a2mvs : ro_ainposition)
           | ro_a2load | ro_a2unload
           | ro_mvto (ro_mvto : ro_ainposition)
           | ro_a1at (ro_a1ats : ro_ainposition)
           | ro_a2at (ro_a2ats : ro_ainposition)
           | ro_at (ro_at : ro_ainposition)
           ;
variables roe1: ro_event; ropo1: ro_ainposition; roap1: ro_ainposition;
end data specification

PRESS_EVENTS =
data specification
pr_ainposition = pr_up | pr_toup
                | pr_down | pr_todown
                | pr_mid | pr_tomid
                ;
pr_event = pr_mvto (pr_mvto : pr_ainposition)
          | pr_load | pr_unload
          | pr_at (pr_at : pr_ainposition)
          ;
variables pre1: pr_event; prpo1: pr_ainposition;
end data specification

DBELT_EVENTS =
data specification
db_event = db_load | db_unload | db_on | db_off
          | db_pbinterruption | db_sheetfalls
          ;
variables dbe1: db_event;
end data specification

CRANE_EVENTS =
data specification
cr_ainposition = cr_aup | cr_atoup
                | cr_adown | cr_atodown
                | cr_amid | cr_atomid
                ;
cr_ainposition = cr_fb | cr_tofb
                | cr_db | cr_todb
                ;
cr_event = cr_amvto (cr_amvs : cr_ainposition)
          | cr_mvto (cr_mvto : cr_ainposition)
          | cr_load | cr_unload
          | cr_aat (cr_aats : cr_ainposition)
          | cr_at (cr_at : cr_ainposition)
          ;
variables hge1: cr_event; hgpo1: cr_ainposition; hgap1: cr_ainposition;
end data specification

```

## A.2 System Independent Specification of Devices

**GENERIC\_FBELT =**  
**enrich FBELT\_EVENTS with**  
**sorts** fb\_state;  
**constants** fberr : fb\_state; fbinit : fb\_state;  
**functions** fb\_trans : fb\_event  $\times$  fb\_state  $\rightarrow$  fb\_state ;  
**predicates**  
     fb\_errp                 : fb\_state;  
     fb\_exp                 : fb\_event  $\times$  fb\_state;  
     fb\_loaded             : fb\_state;  
     fb\_pbinterrupted     : fb\_state;  
     fb\_running            : fb\_state;  
**variables** fb<sub>2</sub>, fb<sub>1</sub>: fb\_state;  
**axioms**  
     fb\_state **generated by** fb\_trans, fbinit;  
  
     fb\_errp(fb<sub>1</sub>)  $\leftrightarrow$  fb<sub>1</sub> = fberr,  
     fbinit  $\neq$  fberr,  
     fb\_pbinterrupted(fb<sub>1</sub>)  $\wedge$   $\neg$  fb\_loaded(fb<sub>1</sub>)  $\rightarrow$  fb<sub>1</sub> = fberr,  
  
     fb<sub>1</sub>  $\neq$  fberr  
      $\rightarrow$  (fb<sub>1</sub> = fb<sub>2</sub>  
          $\leftrightarrow$  fb<sub>2</sub>  $\neq$  fberr  
          $\wedge$  (fb\_running(fb<sub>1</sub>)  $\leftrightarrow$  fb\_running(fb<sub>2</sub>))  
          $\wedge$  (fb\_loaded(fb<sub>1</sub>)  $\leftrightarrow$  fb\_loaded(fb<sub>2</sub>))  
          $\wedge$  (fb\_pbinterrupted(fb<sub>1</sub>)  $\leftrightarrow$  fb\_pbinterrupted(fb<sub>2</sub>)) ),  
  
     fb\_exp(fb\_sheetfalls, fb<sub>1</sub>)  
      $\leftrightarrow$  fb<sub>1</sub>  $\neq$  fberr  $\wedge$  fb\_running(fb<sub>1</sub>)  $\wedge$  fb\_loaded(fb<sub>1</sub>)  $\wedge$  fb\_pbinterrupted(fb<sub>1</sub>),  
     fb\_exp(fb\_pbinterruption, fb<sub>1</sub>)  
      $\leftrightarrow$  fb<sub>1</sub>  $\neq$  fberr  $\wedge$  fb\_running(fb<sub>1</sub>)  $\wedge$  fb\_loaded(fb<sub>1</sub>)  $\wedge$   $\neg$  fb\_pbinterrupted(fb<sub>1</sub>),  
  
      $\neg$  fb\_exp(fb\_on, fb<sub>1</sub>),  
      $\neg$  fb\_exp(fb\_off, fb<sub>1</sub>),  
      $\neg$  fb\_exp(fb\_load, fb<sub>1</sub>),  
      $\neg$  fb\_exp(fb\_unload, fb<sub>1</sub>),  
  
     fb<sub>1</sub>  $\neq$  fberr  $\wedge$   $\neg$  fb\_running(fb<sub>1</sub>)  $\wedge$  fb<sub>2</sub> = fb\_trans(fb\_on, fb<sub>1</sub>)  
      $\rightarrow$  fb<sub>2</sub>  $\neq$  fberr  $\wedge$  fb\_running(fb<sub>2</sub>)  $\wedge$   
         (fb\_loaded(fb<sub>2</sub>)  $\leftrightarrow$  fb\_loaded(fb<sub>1</sub>))  $\wedge$   
         (fb\_pbinterrupted(fb<sub>2</sub>)  $\leftrightarrow$  fb\_pbinterrupted(fb<sub>1</sub>)),  
  
     fb<sub>1</sub> = fberr  $\vee$  fb\_running(fb<sub>1</sub>)  
      $\rightarrow$  fb\_trans(fb\_on, fb<sub>1</sub>) = fberr,  
  
     fb<sub>1</sub>  $\neq$  fberr  $\wedge$  fb\_running(fb<sub>1</sub>)  $\wedge$  fb<sub>2</sub> = fb\_trans(fb\_off, fb<sub>1</sub>)  
      $\rightarrow$  fb<sub>2</sub>  $\neq$  fberr  $\wedge$   $\neg$  fb\_running(fb<sub>2</sub>)  $\wedge$   
         (fb\_loaded(fb<sub>2</sub>)  $\leftrightarrow$  fb\_loaded(fb<sub>1</sub>))  $\wedge$   
         (fb\_pbinterrupted(fb<sub>2</sub>)  $\leftrightarrow$  fb\_pbinterrupted(fb<sub>1</sub>)),  
  
     fb<sub>1</sub> = fberr  $\vee$   $\neg$  fb\_running(fb<sub>1</sub>)  
      $\rightarrow$  fb\_trans(fb\_off, fb<sub>1</sub>) = fberr,



$$\begin{aligned}
& fb_1 \neq fberr \wedge fb\_running(fb_1) \wedge \\
& fb\_loaded(fb_1) \wedge \neg fb\_pbinterrupted(fb_1) \wedge fb_2 = fb\_trans(fb\_pbinterruption, fb_1) \\
\rightarrow & fb_2 \neq fberr \wedge fb\_running(fb_2) \wedge fb\_loaded(fb_2) \wedge fb\_pbinterrupted(fb_2), \\
\\
& fb_1 = fberr \vee \neg fb\_running(fb_1) \vee \neg fb\_loaded(fb_1) \vee fb\_pbinterrupted(fb_1) \\
\rightarrow & fb\_trans(fb\_pbinterruption, fb_1) = fberr, \\
\\
& fb_1 \neq fberr \wedge fb\_running(fb_1) \wedge fb\_loaded(fb_1) \wedge \\
& fb\_pbinterrupted(fb_1) \wedge fb_2 = fb\_trans(fb\_sheetfalls, fb_1) \\
\rightarrow & fb_2 \neq fberr \wedge fb\_running(fb_2) \wedge \neg fb\_loaded(fb_2) \wedge \neg fb\_pbinterrupted(fb_2), \\
\\
& fb_1 = fberr \vee \neg fb\_running(fb_1) \vee \neg fb\_loaded(fb_1) \vee \neg fb\_pbinterrupted(fb_1) \\
\rightarrow & fb\_trans(fb\_sheetfalls, fb_1) = fberr, \\
\\
& fb_1 \neq fberr \wedge \neg fb\_loaded(fb_1) \wedge fb_2 = fb\_trans(fb\_load, fb_1) \\
\rightarrow & fb_2 \neq fberr \wedge (fb\_running(fb_2) \leftrightarrow fb\_running(fb_1)) \wedge \\
& fb\_loaded(fb_2) \wedge \neg fb\_pbinterrupted(fb_2), \\
\\
& fb_1 = fberr \vee fb\_loaded(fb_1) \\
\rightarrow & fb\_trans(fb\_load, fb_1) = fberr, \\
\\
& fb_1 \neq fberr \wedge fb\_loaded(fb_1) \wedge \\
& \neg fb\_running(fb_1) \wedge fb\_pbinterrupted(fb_1) \wedge fb_2 = fb\_trans(fb\_unload, fb_1) \\
\rightarrow & fb_2 \neq fberr \wedge \neg fb\_running(fb_2) \wedge \neg fb\_loaded(fb_2) \wedge \neg fb\_pbinterrupted(fb_2), \\
\\
& fb_1 = fberr \vee fb\_running(fb_1) \vee \neg fb\_loaded(fb_1) \vee \neg fb\_pbinterrupted(fb_1) \\
\rightarrow & fb\_trans(fb\_unload, fb_1) = fberr
\end{aligned}$$

**end enrich**

GENERIC\_TABLE =

**enrich** TABLE\_EVENTS **with**

**sorts** ta\_state;

**constants** taerr : ta\_state; tainit : ta\_state;

**functions**

ta\_pos : ta\_state  $\rightarrow$  ta\_position ;

ta\_tpos : ta\_state  $\rightarrow$  ta\_tposition ;

ta\_trans : ta\_event  $\times$  ta\_state  $\rightarrow$  ta\_state ;

**predicates**

ta\_errp : ta\_state;

ta\_exp : ta\_event  $\times$  ta\_state;

ta\_loaded : ta\_state;

**variables** ta<sub>2</sub>, ta<sub>1</sub>: ta\_state; ta<sub>e1</sub>: ta\_event;

**axioms**

ta\_state **generated by** ta\_trans, tainit;

ta\_errp(ta<sub>1</sub>)  $\leftrightarrow$  ta<sub>1</sub> = taerr,

tainit  $\neq$  taerr,

ta\_pos(ta<sub>1</sub>)  $\neq$  ta<sub>up</sub>  $\wedge$  ta\_tpos(ta<sub>1</sub>)  $\neq$  ta\_fb  $\rightarrow$  ta<sub>1</sub> = taerr,

$$\begin{aligned}
& ta_1 \neq taerr \\
\rightarrow & (ta_1 = ta_2 \\
\leftrightarrow & ta_2 \neq taerr \\
& \wedge ta\_pos(ta_1) = ta\_pos(ta_2) \\
& \wedge ta\_tpos(ta_1) = ta\_tpos(ta_2) \\
& \wedge (ta\_loaded(ta_1) \leftrightarrow ta\_loaded(ta_2)) ),
\end{aligned}$$

$$\begin{aligned}
& ta\_exp(ta\_at(ta\_up), ta_1) \\
\leftrightarrow & ta_1 \neq taerr \wedge ta\_pos(ta_1) = ta\_toup, \\
& ta\_exp(ta\_at(ta\_down), ta_1) \\
\leftrightarrow & ta_1 \neq taerr \wedge ta\_pos(ta_1) = ta\_todown, \\
& ta\_exp(ta\_tat(ta\_fb), ta_1) \\
\leftrightarrow & ta_1 \neq taerr \wedge ta\_tpos(ta_1) = ta\_tofb, \\
& ta\_exp(ta\_tat(ta\_ro), ta_1) \\
\leftrightarrow & ta_1 \neq taerr \wedge ta\_tpos(ta_1) = ta\_toro,
\end{aligned}$$

$$\begin{aligned}
& \neg ta\_exp(ta\_mvto(tapo_1), ta_1), \\
& \neg ta\_exp(ta\_tmvto(tatpo_1), ta_1), \\
& \neg ta\_exp(ta\_load, ta_1), \\
& \neg ta\_exp(ta\_unload, ta_1), \\
& \neg ta\_exp(ta\_at(ta\_toup), ta_1), \\
& \neg ta\_exp(ta\_at(ta\_todown), ta_1), \\
& \neg ta\_exp(ta\_tat(ta\_tofb), ta_1), \\
& \neg ta\_exp(ta\_tat(ta\_toro), ta_1),
\end{aligned}$$

$$\begin{aligned}
& ta_1 \neq taerr \wedge ta\_pos(ta_1) = ta\_down \wedge \\
& ta\_tpos(ta_1) = ta\_fb \wedge ta_2 = ta\_trans(ta\_mvto(ta\_up), ta_1) \\
\rightarrow & ta_2 \neq taerr \wedge ta\_pos(ta_2) = ta\_toup \wedge \\
& ta\_tpos(ta_2) = ta\_fb \wedge (ta\_loaded(ta_2) \leftrightarrow ta\_loaded(ta_1)),
\end{aligned}$$

$$\begin{aligned}
& ta_1 = taerr \vee ta\_pos(ta_1) \neq ta\_down \vee ta\_tpos(ta_1) \neq ta\_fb \\
\rightarrow & ta\_trans(ta\_mvto(ta\_up), ta_1) = taerr,
\end{aligned}$$

$$\begin{aligned}
& ta_1 \neq taerr \wedge ta\_pos(ta_1) = ta\_up \wedge \\
& ta\_tpos(ta_1) = ta\_fb \wedge ta_2 = ta\_trans(ta\_mvto(ta\_down), ta_1) \\
\rightarrow & ta_2 \neq taerr \wedge ta\_pos(ta_2) = ta\_todown \wedge \\
& ta\_tpos(ta_2) = ta\_fb \wedge (ta\_loaded(ta_2) \leftrightarrow ta\_loaded(ta_1)),
\end{aligned}$$

$$\begin{aligned}
& ta_1 = taerr \vee ta\_pos(ta_1) \neq ta\_up \vee ta\_tpos(ta_1) \neq ta\_fb \\
\rightarrow & ta\_trans(ta\_mvto(ta\_down), ta_1) = taerr,
\end{aligned}$$

$$\begin{aligned}
& ta_1 \neq taerr \wedge ta\_pos(ta_1) = ta\_up \wedge \\
& ta\_tpos(ta_1) = ta\_ro \wedge ta_2 = ta\_trans(ta\_tmvto(ta\_fb), ta_1) \\
\rightarrow & ta_2 \neq taerr \wedge ta\_pos(ta_2) = ta\_up \wedge \\
& ta\_tpos(ta_2) = ta\_tofb \wedge (ta\_loaded(ta_2) \leftrightarrow ta\_loaded(ta_1)),
\end{aligned}$$

$$\begin{aligned}
& ta_1 = taerr \vee ta\_pos(ta_1) \neq ta\_up \vee ta\_tpos(ta_1) \neq ta\_ro \\
\rightarrow & ta\_trans(ta\_tmvto(ta\_fb), ta_1) = taerr,
\end{aligned}$$

$$\begin{aligned}
& ta_1 \neq taerr \wedge ta\_pos(ta_1) = ta\_up \wedge \\
& ta\_tpos(ta_1) = ta\_fb \wedge ta_2 = ta\_trans(ta\_tmvto(ta\_ro), ta_1) \\
\rightarrow & ta_2 \neq taerr \wedge ta\_pos(ta_2) = ta\_up \wedge \\
& ta\_tpos(ta_2) = ta\_toro \wedge (ta\_loaded(ta_2) \leftrightarrow ta\_loaded(ta_1)),
\end{aligned}$$

$ta_1 = ta_{err} \vee ta_{pos}(ta_1) \neq ta_{up} \vee ta_{tpos}(ta_1) \neq ta_{fb}$   
 $\rightarrow ta_{trans}(ta_{tmvto}(ta_{ro}), ta_1) = ta_{err},$

$ta_1 \neq ta_{err} \wedge ta_{pos}(ta_1) = ta_{toup} \wedge$   
 $ta_{tpos}(ta_1) = ta_{fb} \wedge ta_2 = ta_{trans}(ta_{at}(ta_{up}), ta_1)$   
 $\rightarrow ta_2 \neq ta_{err} \wedge ta_{pos}(ta_2) = ta_{up} \wedge$   
 $ta_{tpos}(ta_2) = ta_{fb} \wedge (ta_{loaded}(ta_2) \leftrightarrow ta_{loaded}(ta_1)),$

$ta_1 = ta_{err} \vee ta_{pos}(ta_1) \neq ta_{toup} \vee ta_{tpos}(ta_1) \neq ta_{fb}$   
 $\rightarrow ta_{trans}(ta_{at}(ta_{up}), ta_1) = ta_{err},$

$ta_1 \neq ta_{err} \wedge ta_{pos}(ta_1) = ta_{todown} \wedge$   
 $ta_{tpos}(ta_1) = ta_{fb} \wedge ta_2 = ta_{trans}(ta_{at}(ta_{down}), ta_1)$   
 $\rightarrow ta_2 \neq ta_{err} \wedge ta_{pos}(ta_2) = ta_{down} \wedge$   
 $ta_{tpos}(ta_2) = ta_{fb} \wedge (ta_{loaded}(ta_2) \leftrightarrow ta_{loaded}(ta_1)),$

$ta_1 = ta_{err} \vee ta_{pos}(ta_1) \neq ta_{todown} \vee ta_{tpos}(ta_1) \neq ta_{fb}$   
 $\rightarrow ta_{trans}(ta_{at}(ta_{down}), ta_1) = ta_{err},$

$ta_1 \neq ta_{err} \wedge ta_{pos}(ta_1) = ta_{up} \wedge$   
 $ta_{tpos}(ta_1) = ta_{tofb} \wedge ta_2 = ta_{trans}(ta_{tat}(ta_{fb}), ta_1)$   
 $\rightarrow ta_2 \neq ta_{err} \wedge ta_{pos}(ta_2) = ta_{up} \wedge$   
 $ta_{tpos}(ta_2) = ta_{fb} \wedge (ta_{loaded}(ta_2) \leftrightarrow ta_{loaded}(ta_1)),$

$ta_1 = ta_{err} \vee ta_{pos}(ta_1) \neq ta_{up} \vee ta_{tpos}(ta_1) \neq ta_{tofb}$   
 $\rightarrow ta_{trans}(ta_{tat}(ta_{fb}), ta_1) = ta_{err},$

$ta_1 \neq ta_{err} \wedge ta_{pos}(ta_1) = ta_{up} \wedge$   
 $ta_{tpos}(ta_1) = ta_{toro} \wedge ta_2 = ta_{trans}(ta_{tat}(ta_{ro}), ta_1)$   
 $\rightarrow ta_2 \neq ta_{err} \wedge ta_{pos}(ta_2) = ta_{up} \wedge$   
 $ta_{tpos}(ta_2) = ta_{ro} \wedge (ta_{loaded}(ta_2) \leftrightarrow ta_{loaded}(ta_1)),$

$ta_1 = ta_{err} \vee ta_{pos}(ta_1) \neq ta_{up} \vee ta_{tpos}(ta_1) \neq ta_{toro}$   
 $\rightarrow ta_{trans}(ta_{tat}(ta_{ro}), ta_1) = ta_{err},$

$ta_1 \neq ta_{err} \wedge (ta_{pos}(ta_1) = ta_{up} \vee ta_{pos}(ta_1) = ta_{down}) \wedge$   
 $(ta_{tpos}(ta_1) = ta_{ro} \vee ta_{tpos}(ta_1) = ta_{fb}) \wedge$   
 $\neg ta_{loaded}(ta_1) \wedge ta_2 = ta_{trans}(ta_{load}, ta_1)$   
 $\rightarrow ta_2 \neq ta_{err} \wedge ta_{pos}(ta_2) = ta_{pos}(ta_1) \wedge ta_{tpos}(ta_2) = ta_{tpos}(ta_1) \wedge ta_{loaded}(ta_2),$

$ta_1 = ta_{err} \vee ta_{pos}(ta_1) \neq ta_{up} \wedge ta_{pos}(ta_1) \neq ta_{down} \vee$   
 $ta_{tpos}(ta_1) \neq ta_{ro} \wedge ta_{tpos}(ta_1) \neq ta_{fb} \vee ta_{loaded}(ta_1)$   
 $\rightarrow ta_{trans}(ta_{load}, ta_1) = ta_{err},$

$ta_1 \neq ta_{err} \wedge (ta_{pos}(ta_1) = ta_{up} \vee ta_{pos}(ta_1) = ta_{down}) \wedge$   
 $(ta_{tpos}(ta_1) = ta_{ro} \vee ta_{tpos}(ta_1) = ta_{fb}) \wedge$   
 $ta_{loaded}(ta_1) \wedge ta_2 = ta_{trans}(ta_{unload}, ta_1)$   
 $\rightarrow ta_2 \neq ta_{err} \wedge ta_{pos}(ta_2) = ta_{pos}(ta_1) \wedge ta_{tpos}(ta_2) = ta_{tpos}(ta_1) \wedge \neg ta_{loaded}(ta_2),$

$ta_1 = ta_{err} \vee ta_{pos}(ta_1) \neq ta_{up} \wedge ta_{pos}(ta_1) \neq ta_{down} \vee$   
 $ta_{tpos}(ta_1) \neq ta_{ro} \wedge ta_{tpos}(ta_1) \neq ta_{fb} \vee \neg ta_{loaded}(ta_1)$   
 $\rightarrow ta_{trans}(ta_{unload}, ta_1) = ta_{err}$

**end enrich**

```

GENERIC_ROBOT =
enrich ROBOT_EVENTS with
  sorts ro_state;
  constants roerr : ro_state; roinit : ro_state;
  functions
    ro_a1pos  : ro_state      → ro_a_position  ;
    ro_a2pos  : ro_state      → ro_a_position  ;
    ro_pos    : ro_state      → ro_position    ;
    ro_trans  : ro_event × ro_state → ro_state  ;
  predicates
    ro_errp   : ro_state;
    ro_exp    : ro_event × ro_state;
    ro_aloaded : ro_state;
    ro_a2loaded : ro_state;
  variables roe1 : ro_event; ro4, ro3, ro2, ro1, ro0 : ro_state;
  axioms
    ro_state generated by ro_trans, roinit;

    ro_errp(ro1) ↔ ro1 = roerr,
    roinit ≠ roerr,
    (ro_pos(ro1) = ro_alupr ∨ ro_pos(ro1) = ro_a2upr ∨
     ro_pos(ro1) = ro_altota ∨ ro_pos(ro1) = ro_a2todb)
    ∧ (ro_a1pos(ro1) ≠ ro_ain ∨ ro_a2pos(ro1) ≠ ro_ain)
    → ro1 = roerr,

    ro_pos(ro1) ≠ ro_alta ∧ ro_pos(ro1) ≠ ro_alpr ∧ ro_a1pos(ro1) ≠ ro_ain → ro1 = roerr,
    ro_pos(ro1) ≠ ro_a2db ∧ ro_pos(ro1) ≠ ro_a2pr ∧ ro_a2pos(ro1) ≠ ro_ain → ro1 = roerr,

    ro1 ≠ roerr
  → (ro1 = ro2
     ↔ ro2 ≠ roerr
        ∧ (ro_aloaded(ro1) ↔ ro_aloaded(ro2))
        ∧ (ro_a2loaded(ro1) ↔ ro_a2loaded(ro2))
        ∧ ro_pos(ro1) = ro_pos(ro2)
        ∧ ro_a1pos(ro1) = ro_a1pos(ro2)
        ∧ ro_a2pos(ro1) = ro_a2pos(ro2) ),

    ro_exp(ro_alat(ro_ain), ro1) ↔ ro1 ≠ roerr ∧ ro_a1pos(ro1) = ro_atoin,
    ro_exp(ro_a2at(ro_ain), ro1) ↔ ro1 ≠ roerr ∧ ro_a2pos(ro1) = ro_atoin,
    ro_exp(ro_alat(ro_amid), ro1) ↔ ro1 ≠ roerr ∧ ro_a1pos(ro1) = ro_atomid,
    ro_exp(ro_a2at(ro_amid), ro1) ↔ ro1 ≠ roerr ∧ ro_a2pos(ro1) = ro_atomid,
    ro_exp(ro_alat(ro_aout), ro1) ↔ ro1 ≠ roerr ∧ ro_a1pos(ro1) = ro_atoout,
    ro_exp(ro_a2at(ro_aout), ro1) ↔ ro1 ≠ roerr ∧ ro_a2pos(ro1) = ro_atoout,
    ro_exp(ro_at(ro_alta), ro1) ↔ ro1 ≠ roerr ∧ ro_pos(ro1) = ro_altota,
    ro_exp(ro_at(ro_alpr), ro1) ↔ ro1 ≠ roerr ∧ ro_pos(ro1) = ro_alupr,
    ro_exp(ro_at(ro_a2db), ro1) ↔ ro1 ≠ roerr ∧ ro_pos(ro1) = ro_a2todb,
    ro_exp(ro_at(ro_a2pr), ro1) ↔ ro1 ≠ roerr ∧ ro_pos(ro1) = ro_a2upr,

    ¬ ro_exp(ro_mvto(ro_po1), ro1),
    ¬ ro_exp(ro_alload, ro1),
    ¬ ro_exp(ro_a2load, ro1),
    ¬ ro_exp(ro_alunload, ro1),
    ¬ ro_exp(ro_a2unload, ro1),
    ¬ ro_exp(ro_at(ro_alta), ro1),

```



$ro_1 \neq roerr \wedge$   
 $(ro\_pos(ro_1) = ro\_alta \vee ro\_pos(ro_1) = ro\_a2pr \vee ro\_pos(ro_1) = ro\_a2db) \wedge$   
 $ro\_alpos(ro_1) = ro\_ain \wedge ro\_a2pos(ro_1) = ro\_ain \wedge$   
 $ro_2 = ro\_trans(ro\_mvto(ro\_a1pr), ro_1)$   
 $\rightarrow ro_2 \neq roerr \wedge ro\_pos(ro_2) = ro\_a1upr \wedge ro\_alpos(ro_2) = ro\_ain \wedge$   
 $ro\_a2pos(ro_2) = ro\_ain \wedge (ro\_alloaded(ro_2) \leftrightarrow ro\_alloaded(ro_1)) \wedge$   
 $(ro\_a2loaded(ro_2) \leftrightarrow ro\_a2loaded(ro_1)),$

$ro_1 = roerr \vee$   
 $ro\_pos(ro_1) \neq ro\_alta \wedge ro\_pos(ro_1) \neq ro\_a2pr \wedge ro\_pos(ro_1) \neq ro\_a2db \vee$   
 $ro\_alpos(ro_1) \neq ro\_ain \vee ro\_a2pos(ro_1) \neq ro\_ain$   
 $\rightarrow ro\_trans(ro\_mvto(ro\_a1pr), ro_1) = roerr,$

$ro_1 \neq roerr \wedge$   
 $(ro\_pos(ro_1) = ro\_a1pr \vee ro\_pos(ro_1) = ro\_a2pr \vee ro\_pos(ro_1) = ro\_a2db) \wedge$   
 $ro\_alpos(ro_1) = ro\_ain \wedge ro\_a2pos(ro_1) = ro\_ain \wedge$   
 $ro_2 = ro\_trans(ro\_mvto(ro\_alta), ro_1)$   
 $\rightarrow ro_2 \neq roerr \wedge ro\_pos(ro_2) = ro\_altota \wedge$   
 $ro\_alpos(ro_2) = ro\_ain \wedge ro\_a2pos(ro_2) = ro\_ain \wedge$   
 $(ro\_alloaded(ro_2) \leftrightarrow ro\_alloaded(ro_1)) \wedge$   
 $(ro\_a2loaded(ro_2) \leftrightarrow ro\_a2loaded(ro_1)),$

$ro_1 = roerr \vee$   
 $ro\_pos(ro_1) \neq ro\_a1pr \wedge ro\_pos(ro_1) \neq ro\_a2pr \wedge ro\_pos(ro_1) \neq ro\_a2db \vee$   
 $ro\_alpos(ro_1) \neq ro\_ain \vee ro\_a2pos(ro_1) \neq ro\_ain$   
 $\rightarrow ro\_trans(ro\_mvto(ro\_alta), ro_1) = roerr,$

$ro_1 \neq roerr \wedge$   
 $(ro\_pos(ro_1) = ro\_alta \vee ro\_pos(ro_1) = ro\_a1pr \vee ro\_pos(ro_1) = ro\_a2db) \wedge$   
 $ro\_alpos(ro_1) = ro\_ain \wedge ro\_a2pos(ro_1) = ro\_ain \wedge$   
 $ro_2 = ro\_trans(ro\_mvto(ro\_a2pr), ro_1)$   
 $\rightarrow ro_2 \neq roerr \wedge ro\_pos(ro_2) = ro\_a2upr \wedge$   
 $ro\_alpos(ro_2) = ro\_ain \wedge ro\_a2pos(ro_2) = ro\_ain \wedge$   
 $(ro\_alloaded(ro_2) \leftrightarrow ro\_alloaded(ro_1)) \wedge$   
 $(ro\_a2loaded(ro_2) \leftrightarrow ro\_a2loaded(ro_1)),$

$ro_1 = roerr \vee$   
 $ro\_pos(ro_1) \neq ro\_alta \wedge ro\_pos(ro_1) \neq ro\_a1pr \wedge ro\_pos(ro_1) \neq ro\_a2db \vee$   
 $ro\_alpos(ro_1) \neq ro\_ain \vee ro\_a2pos(ro_1) \neq ro\_ain$   
 $\rightarrow ro\_trans(ro\_mvto(ro\_a2pr), ro_1) = roerr,$

$ro_1 \neq roerr \wedge$   
 $(ro\_pos(ro_1) = ro\_alta \vee ro\_pos(ro_1) = ro\_a1pr \vee ro\_pos(ro_1) = ro\_a2pr) \wedge$   
 $ro\_pos(ro_1) \neq ro\_a2db \wedge ro\_alpos(ro_1) = ro\_ain \wedge ro\_a2pos(ro_1) = ro\_ain \wedge$   
 $ro_2 = ro\_trans(ro\_mvto(ro\_a2db), ro_1)$   
 $\rightarrow ro_2 \neq roerr \wedge ro\_pos(ro_2) = ro\_a2todb \wedge$   
 $ro\_alpos(ro_2) = ro\_ain \wedge ro\_a2pos(ro_2) = ro\_ain \wedge$   
 $(ro\_alloaded(ro_2) \leftrightarrow ro\_alloaded(ro_1)) \wedge$   
 $(ro\_a2loaded(ro_2) \leftrightarrow ro\_a2loaded(ro_1)),$

$ro_1 = roerr \vee$   
 $ro\_pos(ro_1) \neq ro\_alta \wedge ro\_pos(ro_1) \neq ro\_a1pr \wedge ro\_pos(ro_1) \neq ro\_a2pr \vee$   
 $ro\_alpos(ro_1) \neq ro\_ain \vee ro\_a2pos(ro_1) \neq ro\_ain$   
 $\rightarrow ro\_trans(ro\_mvto(ro\_a2db), ro_1) = roerr,$

$$\begin{aligned}
& ro_1 \neq roerr \wedge (ro\_pos(ro_1) = ro\_alta \vee ro\_pos(ro_1) = ro\_alpr) \wedge \\
& ro\_alpos(ro_1) = ro\_atoin \wedge ro_2 = ro\_trans(ro\_alat(ro\_ain), ro_1) \\
\rightarrow & ro_2 \neq roerr \wedge ro\_pos(ro_2) = ro\_pos(ro_1) \wedge \\
& ro\_a2pos(ro_2) = ro\_a2pos(ro_1) \wedge ro\_alpos(ro_2) = ro\_ain \wedge \\
& (ro\_alloaded(ro_2) \leftrightarrow ro\_alloaded(ro_1)) \wedge \\
& (ro\_a2loaded(ro_2) \leftrightarrow ro\_a2loaded(ro_1)),
\end{aligned}$$

$$\begin{aligned}
& ro_1 = roerr \vee ro\_alpos(ro_1) \neq ro\_atoin \vee \\
& ro\_pos(ro_1) \neq ro\_alta \wedge ro\_pos(ro_1) \neq ro\_alpr \\
\rightarrow & ro\_trans(ro\_alat(ro\_ain), ro_1) = roerr,
\end{aligned}$$

$$\begin{aligned}
& ro_1 \neq roerr \wedge (ro\_pos(ro_1) = ro\_alta \vee ro\_pos(ro_1) = ro\_alpr) \wedge \\
& ro\_alpos(ro_1) = ro\_atomid \wedge ro_2 = ro\_trans(ro\_alat(ro\_amid), ro_1) \\
\rightarrow & ro_2 \neq roerr \wedge ro\_pos(ro_2) = ro\_pos(ro_1) \wedge \\
& ro\_a2pos(ro_2) = ro\_a2pos(ro_1) \wedge ro\_alpos(ro_2) = ro\_amid \wedge \\
& (ro\_alloaded(ro_2) \leftrightarrow ro\_alloaded(ro_1)) \wedge \\
& (ro\_a2loaded(ro_2) \leftrightarrow ro\_a2loaded(ro_1)),
\end{aligned}$$

$$\begin{aligned}
& ro_1 = roerr \vee ro\_alpos(ro_1) \neq ro\_atomid \vee \\
& ro\_pos(ro_1) \neq ro\_alta \wedge ro\_pos(ro_1) \neq ro\_alpr \\
\rightarrow & ro\_trans(ro\_alat(ro\_amid), ro_1) = roerr,
\end{aligned}$$

$$\begin{aligned}
& ro_1 \neq roerr \wedge (ro\_pos(ro_1) = ro\_alta \vee ro\_pos(ro_1) = ro\_alpr) \wedge \\
& ro\_alpos(ro_1) = ro\_atoout \wedge ro_2 = ro\_trans(ro\_alat(ro\_aout), ro_1) \\
\rightarrow & ro_2 \neq roerr \wedge ro\_pos(ro_2) = ro\_pos(ro_1) \wedge \\
& ro\_a2pos(ro_2) = ro\_a2pos(ro_1) \wedge ro\_alpos(ro_2) = ro\_aout \wedge \\
& (ro\_alloaded(ro_2) \leftrightarrow ro\_alloaded(ro_1)) \wedge \\
& (ro\_a2loaded(ro_2) \leftrightarrow ro\_a2loaded(ro_1)),
\end{aligned}$$

$$\begin{aligned}
& ro_1 = roerr \vee ro\_alpos(ro_1) \neq ro\_atoout \vee \\
& ro\_pos(ro_1) \neq ro\_alta \wedge ro\_pos(ro_1) \neq ro\_alpr \\
\rightarrow & ro\_trans(ro\_alat(ro\_aout), ro_1) = roerr,
\end{aligned}$$

$$\begin{aligned}
& ro_1 \neq roerr \wedge (ro\_pos(ro_1) = ro\_a2pr \vee ro\_pos(ro_1) = ro\_a2db) \wedge \\
& ro\_a2pos(ro_1) = ro\_atoin \wedge ro_2 = ro\_trans(ro\_a2at(ro\_ain), ro_1) \\
\rightarrow & ro_2 \neq roerr \wedge ro\_pos(ro_2) = ro\_pos(ro_1) \wedge \\
& ro\_alpos(ro_2) = ro\_alpos(ro_1) \wedge ro\_a2pos(ro_2) = ro\_ain \wedge \\
& (ro\_alloaded(ro_2) \leftrightarrow ro\_alloaded(ro_1)) \wedge \\
& (ro\_a2loaded(ro_2) \leftrightarrow ro\_a2loaded(ro_1)),
\end{aligned}$$

$$\begin{aligned}
& ro_1 = roerr \vee ro\_a2pos(ro_1) \neq ro\_atoin \vee \\
& ro\_pos(ro_1) \neq ro\_a2pr \wedge ro\_pos(ro_1) \neq ro\_a2db \\
\rightarrow & ro\_trans(ro\_a2at(ro\_ain), ro_1) = roerr,
\end{aligned}$$

$$\begin{aligned}
& ro_1 \neq roerr \wedge (ro\_pos(ro_1) = ro\_a2pr \vee ro\_pos(ro_1) = ro\_a2db) \wedge \\
& ro\_a2pos(ro_1) = ro\_atomid \wedge \\
& ro_2 = ro\_trans(ro\_a2at(ro\_amid), ro_1) \\
\rightarrow & ro_2 \neq roerr \wedge ro\_pos(ro_2) = ro\_pos(ro_1) \wedge \\
& ro\_alpos(ro_2) = ro\_alpos(ro_1) \wedge ro\_a2pos(ro_2) = ro\_amid \wedge \\
& (ro\_alloaded(ro_2) \leftrightarrow ro\_alloaded(ro_1)) \wedge \\
& (ro\_a2loaded(ro_2) \leftrightarrow ro\_a2loaded(ro_1)),
\end{aligned}$$

$ro_1 = roerr \vee ro\_a2pos(ro_1) \neq ro\_atomid \vee$   
 $ro\_pos(ro_1) \neq ro\_a2pr \wedge ro\_pos(ro_1) \neq ro\_a2db$   
 $\rightarrow ro\_trans(ro\_a2at(ro\_amid), ro_1) = roerr,$

$ro_1 \neq roerr \wedge (ro\_pos(ro_1) = ro\_a2pr \vee ro\_pos(ro_1) = ro\_a2db) \wedge$   
 $ro\_a2pos(ro_1) = ro\_atoout \wedge ro_2 = ro\_trans(ro\_a2at(ro\_aout), ro_1)$   
 $\rightarrow ro_2 \neq roerr \wedge ro\_pos(ro_2) = ro\_pos(ro_1) \wedge$   
 $ro\_alpos(ro_2) = ro\_alpos(ro_1) \wedge ro\_a2pos(ro_2) = ro\_aout \wedge$   
 $(ro\_alloaded(ro_2) \leftrightarrow ro\_alloaded(ro_1)) \wedge$   
 $(ro\_a2loaded(ro_2) \leftrightarrow ro\_a2loaded(ro_1)),$

$ro_1 = roerr \vee ro\_a2pos(ro_1) \neq ro\_atoout \vee$   
 $ro\_pos(ro_1) \neq ro\_a2pr \wedge ro\_pos(ro_1) \neq ro\_a2db$   
 $\rightarrow ro\_trans(ro\_a2at(ro\_aout), ro_1) = roerr,$

$ro_1 \neq roerr \wedge (ro\_pos(ro_1) = ro\_alta \vee ro\_pos(ro_1) = ro\_alpr) \wedge$   
 $(ro\_alpos(ro_1) = ro\_amid \vee ro\_alpos(ro_1) = ro\_aout) \wedge$   
 $ro_2 = ro\_trans(ro\_almv(ro\_ain), ro_1)$   
 $\rightarrow ro_2 \neq roerr \wedge ro\_pos(ro_2) = ro\_pos(ro_1) \wedge$   
 $ro\_a2pos(ro_2) = ro\_a2pos(ro_1) \wedge ro\_alpos(ro_2) = ro\_atoin \wedge$   
 $(ro\_alloaded(ro_2) \leftrightarrow ro\_alloaded(ro_1)) \wedge$   
 $(ro\_a2loaded(ro_2) \leftrightarrow ro\_a2loaded(ro_1)),$

$ro_1 = roerr \vee$   
 $ro\_pos(ro_1) \neq ro\_alta \wedge ro\_pos(ro_1) \neq ro\_alpr \vee$   
 $ro\_alpos(ro_1) \neq ro\_amid \wedge ro\_alpos(ro_1) \neq ro\_aout$   
 $\rightarrow ro\_trans(ro\_almv(ro\_ain), ro_1) = roerr,$

$ro_1 \neq roerr \wedge (ro\_pos(ro_1) = ro\_alta \vee ro\_pos(ro_1) = ro\_alpr) \wedge$   
 $(ro\_alpos(ro_1) = ro\_ain \vee ro\_alpos(ro_1) = ro\_aout) \wedge$   
 $ro_2 = ro\_trans(ro\_almv(ro\_amid), ro_1)$   
 $\rightarrow ro_2 \neq roerr \wedge ro\_pos(ro_2) = ro\_pos(ro_1) \wedge ro\_a2pos(ro_2) = ro\_a2pos(ro_1) \wedge$   
 $ro\_alpos(ro_2) = ro\_atomid \wedge (ro\_alloaded(ro_2) \leftrightarrow ro\_alloaded(ro_1)) \wedge$   
 $(ro\_a2loaded(ro_2) \leftrightarrow ro\_a2loaded(ro_1)),$

$ro_1 = roerr \vee$   
 $ro\_pos(ro_1) \neq ro\_alta \wedge ro\_pos(ro_1) \neq ro\_alpr \vee$   
 $ro\_alpos(ro_1) \neq ro\_ain \wedge ro\_alpos(ro_1) \neq ro\_aout$   
 $\rightarrow ro\_trans(ro\_almv(ro\_amid), ro_1) = roerr,$

$ro_1 \neq roerr \wedge (ro\_pos(ro_1) = ro\_alta \vee ro\_pos(ro_1) = ro\_alpr) \wedge$   
 $(ro\_alpos(ro_1) = ro\_ain \vee ro\_alpos(ro_1) = ro\_amid) \wedge$   
 $ro_2 = ro\_trans(ro\_almv(ro\_aout), ro_1)$   
 $\rightarrow ro_2 \neq roerr \wedge ro\_pos(ro_2) = ro\_pos(ro_1) \wedge ro\_a2pos(ro_2) = ro\_a2pos(ro_1) \wedge$   
 $ro\_alpos(ro_2) = ro\_atoout \wedge (ro\_alloaded(ro_2) \leftrightarrow ro\_alloaded(ro_1)) \wedge$   
 $(ro\_a2loaded(ro_2) \leftrightarrow ro\_a2loaded(ro_1)),$

$ro_1 = roerr \vee$   
 $ro\_pos(ro_1) \neq ro\_alta \wedge ro\_pos(ro_1) \neq ro\_alpr \vee$   
 $ro\_alpos(ro_1) \neq ro\_ain \wedge ro\_alpos(ro_1) \neq ro\_amid$   
 $\rightarrow ro\_trans(ro\_almv(ro\_aout), ro_1) = roerr,$



$$\begin{aligned}
& ro_1 \neq roerr \wedge (ro\_pos(ro_1) = ro\_a2pr \vee ro\_pos(ro_1) = ro\_a2db) \wedge \\
& (ro\_a2pos(ro_1) = ro\_amid \vee ro\_a2pos(ro_1) = ro\_aout) \wedge \\
& ro_2 = ro\_trans(ro\_a2mv(ro\_ain), ro_1) \\
\rightarrow & ro_2 \neq roerr \wedge ro\_pos(ro_2) = ro\_pos(ro_1) \wedge \\
& ro\_a1pos(ro_2) = ro\_a1pos(ro_1) \wedge ro\_a2pos(ro_2) = ro\_atoin \wedge \\
& (ro\_alloaded(ro_2) \leftrightarrow ro\_alloaded(ro_1)) \wedge \\
& (ro\_a2loaded(ro_2) \leftrightarrow ro\_a2loaded(ro_1)),
\end{aligned}$$

$$\begin{aligned}
& ro_1 = roerr \vee \\
& ro\_pos(ro_1) \neq ro\_a2pr \wedge ro\_pos(ro_1) \neq ro\_a2db \vee \\
& ro\_a2pos(ro_1) \neq ro\_amid \wedge ro\_a2pos(ro_1) \neq ro\_aout \\
\rightarrow & ro\_trans(ro\_a2mv(ro\_ain), ro_1) = roerr,
\end{aligned}$$

$$\begin{aligned}
& ro_1 \neq roerr \wedge (ro\_pos(ro_1) = ro\_a2pr \vee ro\_pos(ro_1) = ro\_a2db) \wedge \\
& (ro\_a2pos(ro_1) = ro\_ain \vee ro\_a2pos(ro_1) = ro\_aout) \wedge \\
& ro_2 = ro\_trans(ro\_a2mv(ro\_amid), ro_1) \\
\rightarrow & ro_2 \neq roerr \wedge ro\_pos(ro_2) = ro\_pos(ro_1) \wedge \\
& ro\_a1pos(ro_2) = ro\_a1pos(ro_1) \wedge ro\_a2pos(ro_2) = ro\_atomid \wedge \\
& (ro\_alloaded(ro_2) \leftrightarrow ro\_alloaded(ro_1)) \wedge \\
& (ro\_a2loaded(ro_2) \leftrightarrow ro\_a2loaded(ro_1)),
\end{aligned}$$

$$\begin{aligned}
& ro_1 = roerr \vee \\
& ro\_pos(ro_1) \neq ro\_a2pr \wedge ro\_pos(ro_1) \neq ro\_a2db \vee \\
& ro\_a2pos(ro_1) \neq ro\_ain \wedge ro\_a2pos(ro_1) \neq ro\_aout \\
\rightarrow & ro\_trans(ro\_a2mv(ro\_amid), ro_1) = roerr,
\end{aligned}$$

$$\begin{aligned}
& ro_1 \neq roerr \wedge (ro\_pos(ro_1) = ro\_a2pr \vee ro\_pos(ro_1) = ro\_a2db) \wedge \\
& (ro\_a2pos(ro_1) = ro\_ain \vee ro\_a2pos(ro_1) = ro\_amid) \wedge \\
& ro_2 = ro\_trans(ro\_a2mv(ro\_aout), ro_1) \\
\rightarrow & ro_2 \neq roerr \wedge ro\_pos(ro_2) = ro\_pos(ro_1) \wedge \\
& ro\_a1pos(ro_2) = ro\_a1pos(ro_1) \wedge ro\_a2pos(ro_2) = ro\_atoout \wedge \\
& (ro\_alloaded(ro_2) \leftrightarrow ro\_alloaded(ro_1)) \wedge \\
& (ro\_a2loaded(ro_2) \leftrightarrow ro\_a2loaded(ro_1)),
\end{aligned}$$

$$\begin{aligned}
& ro_1 = roerr \vee \\
& ro\_pos(ro_1) \neq ro\_a2pr \wedge ro\_pos(ro_1) \neq ro\_a2db \vee \\
& ro\_a2pos(ro_1) \neq ro\_ain \wedge ro\_a2pos(ro_1) \neq ro\_amid \\
\rightarrow & ro\_trans(ro\_a2mv(ro\_aout), ro_1) = roerr,
\end{aligned}$$

$$\begin{aligned}
& ro_1 \neq roerr \wedge (ro\_pos(ro_1) = ro\_alta \vee ro\_pos(ro_1) = ro\_alpr) \wedge \\
& (ro\_alpos(ro_1) = ro\_amid \vee ro\_alpos(ro_1) = ro\_aout) \wedge \\
& \neg ro\_alloaded(ro_1) \wedge ro_2 = ro\_trans(ro\_alload, ro_1) \\
\rightarrow & ro_2 \neq roerr \wedge ro\_pos(ro_2) = ro\_pos(ro_1) \wedge \\
& ro\_alpos(ro_2) = ro\_alpos(ro_1) \wedge ro\_a2pos(ro_2) = ro\_a2pos(ro_1) \wedge \\
& ro\_alloaded(ro_2) \wedge (ro\_a2loaded(ro_2) \leftrightarrow ro\_a2loaded(ro_1)),
\end{aligned}$$

$$\begin{aligned}
& ro_1 = roerr \vee \\
& ro\_pos(ro_1) \neq ro\_alta \wedge ro\_pos(ro_1) \neq ro\_alpr \vee \\
& ro\_alpos(ro_1) \neq ro\_amid \wedge ro\_alpos(ro_1) \neq ro\_aout \vee ro\_alloaded(ro_1) \\
\rightarrow & ro\_trans(ro\_alload, ro_1) = roerr,
\end{aligned}$$

$$\begin{aligned}
& ro_1 \neq roerr \wedge (ro\_pos(ro_1) = ro\_alta \vee ro\_pos(ro_1) = ro\_alpr) \wedge \\
& (ro\_alpos(ro_1) = ro\_amid \vee ro\_alpos(ro_1) = ro\_aout) \wedge \\
& ro\_alloaded(ro_1) \wedge ro_2 = ro\_trans(ro\_alunload, ro_1) \\
\rightarrow & ro_2 \neq roerr \wedge ro\_pos(ro_2) = ro\_pos(ro_1) \wedge \\
& ro\_alpos(ro_2) = ro\_alpos(ro_1) \wedge ro\_a2pos(ro_2) = ro\_a2pos(ro_1) \wedge \\
& \neg ro\_alloaded(ro_2) \wedge (ro\_a2loaded(ro_2) \leftrightarrow ro\_a2loaded(ro_1)), \\
& ro_1 = roerr \vee ro\_pos(ro_1) \neq ro\_alta \wedge ro\_pos(ro_1) \neq ro\_alpr \vee \\
& ro\_alpos(ro_1) \neq ro\_amid \wedge ro\_alpos(ro_1) \neq ro\_aout \vee \neg ro\_alloaded(ro_1) \\
\rightarrow & ro\_trans(ro\_alunload, ro_1) = roerr,
\end{aligned}$$

$$\begin{aligned}
& ro_1 \neq roerr \wedge (ro\_pos(ro_1) = ro\_a2db \vee ro\_pos(ro_1) = ro\_a2pr) \wedge \\
& (ro\_a2pos(ro_1) = ro\_amid \vee ro\_a2pos(ro_1) = ro\_aout) \wedge \\
& \neg ro\_a2loaded(ro_1) \wedge ro_2 = ro\_trans(ro\_a2load, ro_1) \\
\rightarrow & ro_2 \neq roerr \wedge ro\_pos(ro_2) = ro\_pos(ro_1) \wedge \\
& ro\_alpos(ro_2) = ro\_alpos(ro_1) \wedge ro\_a2pos(ro_2) = ro\_a2pos(ro_1) \wedge \\
& ro\_a2loaded(ro_2) \wedge (ro\_alloaded(ro_2) \leftrightarrow ro\_alloaded(ro_1)),
\end{aligned}$$

$$\begin{aligned}
& ro_1 = roerr \vee ro\_pos(ro_1) \neq ro\_a2db \wedge ro\_pos(ro_1) \neq ro\_a2pr \vee \\
& ro\_a2pos(ro_1) \neq ro\_amid \wedge ro\_a2pos(ro_1) \neq ro\_aout \vee ro\_a2loaded(ro_1) \\
\rightarrow & ro\_trans(ro\_a2load, ro_1) = roerr,
\end{aligned}$$

$$\begin{aligned}
& ro_1 \neq roerr \wedge (ro\_pos(ro_1) = ro\_a2db \vee ro\_pos(ro_1) = ro\_a2pr) \wedge \\
& (ro\_a2pos(ro_1) = ro\_amid \vee ro\_a2pos(ro_1) = ro\_aout) \wedge \\
& ro\_a2loaded(ro_1) \wedge ro_2 = ro\_trans(ro\_a2unload, ro_1) \\
\rightarrow & ro_2 \neq roerr \wedge ro\_pos(ro_2) = ro\_pos(ro_1) \wedge \\
& ro\_alpos(ro_2) = ro\_alpos(ro_1) \wedge ro\_a2pos(ro_2) = ro\_a2pos(ro_1) \wedge \\
& \neg ro\_a2loaded(ro_2) \wedge (ro\_alloaded(ro_2) \leftrightarrow ro\_alloaded(ro_1)),
\end{aligned}$$

$$\begin{aligned}
& ro_1 = roerr \vee ro\_pos(ro_1) \neq ro\_a2db \wedge ro\_pos(ro_1) \neq ro\_a2pr \vee \\
& ro\_a2pos(ro_1) \neq ro\_amid \wedge ro\_a2pos(ro_1) \neq ro\_aout \vee \neg ro\_a2loaded(ro_1) \\
\rightarrow & ro\_trans(ro\_a2unload, ro_1) = roerr,
\end{aligned}$$

$$\begin{aligned}
& ro\_trans(ro\_at(ro\_altota), ro_1) = roerr, \\
& ro\_trans(ro\_at(ro\_alupr), ro_1) = roerr, \\
& ro\_trans(ro\_at(ro\_a2upr), ro_1) = roerr, \\
& ro\_trans(ro\_at(ro\_a2todb), ro_1) = roerr, \\
& ro\_trans(ro\_mvto(ro\_altota), ro_1) = roerr, \\
& ro\_trans(ro\_mvto(ro\_alupr), ro_1) = roerr, \\
& ro\_trans(ro\_mvto(ro\_a2upr), ro_1) = roerr, \\
& ro\_trans(ro\_mvto(ro\_a2todb), ro_1) = roerr, \\
& ro\_trans(ro\_almv(ro\_atoin), ro_1) = roerr, \\
& ro\_trans(ro\_almv(ro\_atomid), ro_1) = roerr, \\
& ro\_trans(ro\_almv(ro\_atoout), ro_1) = roerr, \\
& ro\_trans(ro\_a2mv(ro\_atoin), ro_1) = roerr, \\
& ro\_trans(ro\_a2mv(ro\_atomid), ro_1) = roerr, \\
& ro\_trans(ro\_a2mv(ro\_atoout), ro_1) = roerr, \\
& ro\_trans(ro\_alat(ro\_atoin), ro_1) = roerr, \\
& ro\_trans(ro\_alat(ro\_atomid), ro_1) = roerr, \\
& ro\_trans(ro\_alat(ro\_atoout), ro_1) = roerr, \\
& ro\_trans(ro\_a2at(ro\_atoin), ro_1) = roerr, \\
& ro\_trans(ro\_a2at(ro\_atomid), ro_1) = roerr, \\
& ro\_trans(ro\_a2at(ro\_atoout), ro_1) = roerr
\end{aligned}$$

**end enrich**

```

GENERIC_PRESS =
enrich PRESS_EVENTS with
  sorts pr_state;
  constants prerr : pr_state; prinit : pr_state;
  functions
    pr_pos    : pr_state          → pr_position  ;
    pr_trans  : pr_event × pr_state → pr_state    ;
  predicates
    pr_errp   : pr_state;
    pr_exp    : pr_event × pr_state;
    pr_pressed : pr_state;
    pr_loaded  : pr_state;
  variables pre2 : pr_event; pr3, pr2, pr1 : pr_state;
  axioms
    pr_state generated by pr_trans, prinit;

    pr_errp(pr1) ↔ pr1 = prerr,
    prinit ≠ prerr,

    pr1 ≠ prerr
  → (pr1 = pr2
    ↔ pr2 ≠ prerr
    ∧ (pr_pressed(pr1) ↔ pr_pressed(pr2))
    ∧ (pr_loaded(pr1) ↔ pr_loaded(pr2))
    ∧ pr_pos(pr1) = pr_pos(pr2) ),

    pr_exp(pr_at(pr_up), pr1) ↔ pr1 ≠ prerr ∧ pr_pos(pr1) = pr_toup,
    pr_exp(pr_at(pr_mid), pr1) ↔ pr1 ≠ prerr ∧ pr_pos(pr1) = pr_tomid,
    pr_exp(pr_at(pr_down), pr1) ↔ pr1 ≠ prerr ∧ pr_pos(pr1) = pr_todown,

    ¬ pr_exp(pr_mvto(prpo1), pr1),
    ¬ pr_exp(pr_load, pr1),
    ¬ pr_exp(pr_unload, pr1),
    ¬ pr_exp(pr_at(pr_toup), pr1),
    ¬ pr_exp(pr_at(pr_tomid), pr1),
    ¬ pr_exp(pr_at(pr_todown), pr1),

    pr1 ≠ prerr ∧ pr_pos(pr1) = pr_toup ∧
    pr_loaded(pr1) ∧ pr2 = pr_trans(pr_at(pr_up), pr1)
  → pr2 ≠ prerr ∧ pr_pos(pr2) = pr_up ∧ pr_loaded(pr2) ∧ pr_pressed(pr2),

    pr1 = prerr ∨ pr_pos(pr1) ≠ pr_toup ∨ ¬ pr_loaded(pr1)
  → pr_trans(pr_at(pr_up), pr1) = prerr,

    pr1 ≠ prerr ∧ pr_pos(pr1) = pr_todown ∧ pr2 = pr_trans(pr_at(pr_down), pr1)
  → pr2 ≠ prerr ∧ pr_pos(pr2) = pr_down ∧
    (pr_loaded(pr2) ↔ pr_loaded(pr1)) ∧ (pr_pressed(pr2) ↔ pr_pressed(pr1)),

    pr1 = prerr ∨ pr_pos(pr1) ≠ pr_todown
  → pr_trans(pr_at(pr_down), pr1) = prerr,

    pr1 ≠ prerr ∧ pr_pos(pr1) = pr_tomid ∧ pr2 = pr_trans(pr_at(pr_mid), pr1)
  → pr2 ≠ prerr ∧ pr_pos(pr2) = pr_mid ∧
    (pr_loaded(pr2) ↔ pr_loaded(pr1)) ∧ (pr_pressed(pr2) ↔ pr_pressed(pr1)),

```

$pr_1 = prerr \vee pr\_pos(pr_1) \neq pr\_tomid$   
 $\rightarrow pr\_trans(pr\_at(pr\_mid), pr_1) = prerr,$

$pr_1 \neq prerr \wedge (pr\_pos(pr_1) = pr\_mid \vee pr\_pos(pr_1) = pr\_down) \wedge$   
 $\neg pr\_loaded(pr_1) \wedge pr_2 = pr\_trans(pr\_load, pr_1)$   
 $\rightarrow pr_2 \neq prerr \wedge pr\_pos(pr_2) = pr\_mid \wedge pr\_loaded(pr_2) \wedge \neg pr\_pressed(pr_2),$

$pr_1 = prerr \vee pr\_pos(pr_1) \neq pr\_mid \wedge pr\_pos(pr_1) \neq pr\_down \vee pr\_loaded(pr_1)$   
 $\rightarrow pr\_trans(pr\_load, pr_1) = prerr,$

$pr_1 \neq prerr \wedge (pr\_pos(pr_1) = pr\_mid \vee pr\_pos(pr_1) = pr\_down) \wedge$   
 $pr\_loaded(pr_1) \wedge pr\_pressed(pr_1) \wedge pr_2 = pr\_trans(pr\_unload, pr_1)$   
 $\rightarrow pr_2 \neq prerr \wedge pr\_pos(pr_2) = pr\_down \wedge \neg pr\_loaded(pr_2),$

$pr_1 = prerr \vee pr\_pos(pr_1) \neq pr\_mid \wedge pr\_pos(pr_1) \neq pr\_down \vee$   
 $\neg pr\_loaded(pr_1) \vee \neg pr\_pressed(pr_1)$   
 $\rightarrow pr\_trans(pr\_unload, pr_1) = prerr,$

$pr_1 \neq prerr \wedge (pr\_pos(pr_1) = pr\_mid \vee pr\_pos(pr_1) = pr\_down) \wedge$   
 $pr_2 = pr\_trans(pr\_mvto(pr\_up), pr_1)$   
 $\rightarrow pr_2 \neq prerr \wedge pr\_pos(pr_2) = pr\_toup \wedge$   
 $(pr\_loaded(pr_2) \leftrightarrow pr\_loaded(pr_1)) \wedge (pr\_pressed(pr_2) \leftrightarrow pr\_pressed(pr_1)),$

$pr_1 = prerr \vee pr\_pos(pr_1) \neq pr\_down \wedge pr\_pos(pr_1) \neq pr\_mid$   
 $\rightarrow pr\_trans(pr\_mvto(pr\_up), pr_1) = prerr,$

$pr_1 \neq prerr \wedge (pr\_pos(pr_1) = pr\_up \vee pr\_pos(pr_1) = pr\_down) \wedge$   
 $pr_2 = pr\_trans(pr\_mvto(pr\_mid), pr_1)$   
 $\rightarrow pr_2 \neq prerr \wedge pr\_pos(pr_2) = pr\_tomid \wedge$   
 $(pr\_loaded(pr_2) \leftrightarrow pr\_loaded(pr_1)) \wedge (pr\_pressed(pr_2) \leftrightarrow pr\_pressed(pr_1)),$

$pr_1 = prerr \vee pr\_pos(pr_1) \neq pr\_up \wedge pr\_pos(pr_1) \neq pr\_down$   
 $\rightarrow pr\_trans(pr\_mvto(pr\_mid), pr_1) = prerr,$

$pr_1 \neq prerr \wedge (pr\_pos(pr_1) = pr\_up \vee pr\_pos(pr_1) = pr\_mid) \wedge$   
 $pr_2 = pr\_trans(pr\_mvto(pr\_down), pr_1)$   
 $\rightarrow pr_2 \neq prerr \wedge pr\_pos(pr_2) = pr\_todown \wedge$   
 $(pr\_loaded(pr_2) \leftrightarrow pr\_loaded(pr_1)) \wedge (pr\_pressed(pr_2) \leftrightarrow pr\_pressed(pr_1)),$

$pr_1 = prerr \vee pr\_pos(pr_1) \neq pr\_up \wedge pr\_pos(pr_1) \neq pr\_mid$   
 $\rightarrow pr\_trans(pr\_mvto(pr\_down), pr_1) = prerr,$

$pr\_trans(pr\_mvto(pr\_toup), pr_1) = prerr,$   
 $pr\_trans(pr\_mvto(pr\_tomid), pr_1) = prerr,$   
 $pr\_trans(pr\_mvto(pr\_todown), pr_1) = prerr,$   
 $pr\_trans(pr\_at(pr\_toup), pr_1) = prerr,$   
 $pr\_trans(pr\_at(pr\_tomid), pr_1) = prerr,$   
 $pr\_trans(pr\_at(pr\_todown), pr_1) = prerr$

**end enrich**

```

GENERIC_DBELT =
enrich DBELT_EVENTS with
  sorts db_state;
  constants dberr : db_state; dbinit : db_state;
  functions db_trans : db_event × db_state → db_state ;
  predicates
    db_errp      : db_state;
    db_exp       : db_event × db_state;
    db_loaded    : db_state;
    db_pbinterrupted : db_state;
    db_running   : db_state;
  variables db2, db1: db_state;
  axioms
    db_state generated by db_trans, dbinit;

    db_errp(db1) ↔ db1 = dberr,
    dbinit ≠ dberr,
    db_pbinterrupted(db1) ∧ ¬ db_loaded(db1) → db1 = dberr,

    db1 ≠ dberr
  → (db1 = db2
    ↔ db2 ≠ dberr
      ∧ (db_running(db1) ↔ db_running(db2))
      ∧ (db_loaded(db1) ↔ db_loaded(db2))
      ∧ (db_pbinterrupted(db1) ↔ db_pbinterrupted(db2))),

    db_exp(db_sheetfalls, db1)
  ↔ db1 ≠ dberr ∧ db_running(db1) ∧ db_loaded(db1) ∧ db_pbinterrupted(db1),
    db_exp(db_pbinterruption, db1)
  ↔ db1 ≠ dberr ∧ db_running(db1) ∧ db_loaded(db1) ∧ ¬ db_pbinterrupted(db1),

    ¬ db_exp(db_on, db1),
    ¬ db_exp(db_off, db1),
    ¬ db_exp(db_load, db1),
    ¬ db_exp(db_unload, db1),

    db1 ≠ dberr ∧ ¬ db_running(db1) ∧ db2 = db_trans(db_on, db1)
  → db2 ≠ dberr ∧ db_running(db2) ∧
    (db_loaded(db2) ↔ db_loaded(db1)) ∧
    (db_pbinterrupted(db2) ↔ db_pbinterrupted(db1)),

    db1 = dberr ∨ db_running(db1)
  → db_trans(db_on, db1) = dberr,

    db1 ≠ dberr ∧ db_running(db1) ∧ db2 = db_trans(db_off, db1)
  → db2 ≠ dberr ∧ ¬ db_running(db2) ∧
    (db_loaded(db2) ↔ db_loaded(db1)) ∧
    (db_pbinterrupted(db2) ↔ db_pbinterrupted(db1)),

    db1 = dberr ∨ ¬ db_running(db1)
  → db_trans(db_off, db1) = dberr,

```

$$\begin{aligned}
& db_1 \neq dberr \wedge db\_running(db_1) \wedge \\
& db\_loaded(db_1) \wedge \neg db\_pbinterrupted(db_1) \wedge db_2 = db\_trans(db\_pbinterruption, db_1) \\
\rightarrow & db_2 \neq dberr \wedge db\_running(db_2) \wedge db\_loaded(db_2) \wedge db\_pbinterrupted(db_2), \\
\\
& db_1 = dberr \vee \neg db\_running(db_1) \vee \neg db\_loaded(db_1) \vee db\_pbinterrupted(db_1) \\
\rightarrow & db\_trans(db\_pbinterruption, db_1) = dberr, \\
\\
& db_1 \neq dberr \wedge db\_running(db_1) \wedge db\_loaded(db_1) \wedge \\
& db\_pbinterrupted(db_1) \wedge db_2 = db\_trans(db\_sheetfalls, db_1) \\
\rightarrow & db_2 \neq dberr \wedge db\_running(db_2) \wedge \neg db\_loaded(db_2) \wedge \neg db\_pbinterrupted(db_2), \\
\\
& db_1 = dberr \vee \neg db\_running(db_1) \vee \neg db\_loaded(db_1) \vee \neg db\_pbinterrupted(db_1) \\
\rightarrow & db\_trans(db\_sheetfalls, db_1) = dberr, \\
\\
& db_1 \neq dberr \wedge \neg db\_loaded(db_1) \wedge db_2 = db\_trans(db\_load, db_1) \\
\rightarrow & db_2 \neq dberr \wedge (db\_running(db_2) \leftrightarrow db\_running(db_1)) \wedge \\
& db\_loaded(db_2) \wedge \neg db\_pbinterrupted(db_2), \\
\\
& db_1 = dberr \vee db\_loaded(db_1) \\
\rightarrow & db\_trans(db\_load, db_1) = dberr, \\
\\
& db_1 \neq dberr \wedge db\_loaded(db_1) \wedge \\
& \neg db\_running(db_1) \wedge db\_pbinterrupted(db_1) \wedge db_2 = db\_trans(db\_unload, db_1) \\
\rightarrow & db_2 \neq dberr \wedge \neg db\_running(db_2) \wedge \neg db\_loaded(db_2) \wedge \neg db\_pbinterrupted(db_2), \\
\\
& db_1 = dberr \vee db\_running(db_1) \vee \neg db\_loaded(db_1) \vee \neg db\_pbinterrupted(db_1) \\
\rightarrow & db\_trans(db\_unload, db_1) = dberr
\end{aligned}$$

**end enrich**

GENERIC\_CRANE =

**enrich CRANE\_EVENTS with**

**sorts** cr\_state;

**constants** crerr : cr\_state; crinit : cr\_state;

**functions**

cr\_apos : cr\_state  $\rightarrow$  cr\_position ;  
cr\_pos : cr\_state  $\rightarrow$  cr\_position ;  
cr\_trans : cr\_event  $\times$  cr\_state  $\rightarrow$  cr\_state ;

**predicates**

cr\_errp : cr\_state;  
cr\_exp : cr\_event  $\times$  cr\_state;  
cr\_aloaded : cr\_state;

**variables** cre1 : cr\_event; cr3, cr2, cr1, cr0 : cr\_state;

**axioms**

cr\_state **generated by** cr\_trans, crinit;

cr\_errp(cr1)  $\leftrightarrow$  cr1 = crerr,  
crinit  $\neq$  crerr,

cr\_apos(cr1)  $\neq$  cr\_aup  $\wedge$  (cr\_pos(cr1) = cr\_tofb  $\vee$  cr\_pos(cr1) = cr\_todb)  
 $\rightarrow$  cr1 = crerr,

$$\begin{aligned}
& cr_1 \neq cr_{err} \\
\rightarrow & (cr_1 = cr_2 \\
& \leftrightarrow cr_2 \neq cr_{err} \\
& \wedge cr\_pos(cr_1) = cr\_pos(cr_2) \\
& \wedge cr\_apos(cr_1) = cr\_apos(cr_2) \\
& \wedge (cr\_aloaded(cr_1) \leftrightarrow cr\_aloaded(cr_2)) ),
\end{aligned}$$

$$\begin{aligned}
cr\_exp(cr\_aat(cr\_aup), cr_1) & \leftrightarrow cr_1 \neq cr_{err} \wedge cr\_apos(cr_1) = cr\_atoup, \\
cr\_exp(cr\_aat(cr\_amid), cr_1) & \leftrightarrow cr_1 \neq cr_{err} \wedge cr\_apos(cr_1) = cr\_atomid, \\
cr\_exp(cr\_aat(cr\_adown), cr_1) & \leftrightarrow cr_1 \neq cr_{err} \wedge cr\_apos(cr_1) = cr\_atodown, \\
cr\_exp(cr\_at(cr\_db), cr_1) & \leftrightarrow cr_1 \neq cr_{err} \wedge cr\_pos(cr_1) = cr\_todb, \\
cr\_exp(cr\_at(cr\_fb), cr_1) & \leftrightarrow cr_1 \neq cr_{err} \wedge cr\_pos(cr_1) = cr\_tofb,
\end{aligned}$$

$$\begin{aligned}
& \neg cr\_exp(cr\_mvto(crpo_1), cr_1), \\
& \neg cr\_exp(cr\_amvto(crap_1), cr_1), \\
& \neg cr\_exp(cr\_load, cr_1), \\
& \neg cr\_exp(cr\_unload, cr_1), \\
& \neg cr\_exp(cr\_aat(cr\_atoup), cr_1), \\
& \neg cr\_exp(cr\_aat(cr\_atomid), cr_1), \\
& \neg cr\_exp(cr\_aat(cr\_atodown), cr_1),
\end{aligned}$$

$$\begin{aligned}
& cr_1 \neq cr_{err} \wedge cr\_apos(cr_1) = cr\_aup \wedge cr\_pos(cr_1) = cr\_db \wedge \\
& cr_2 = cr\_trans(cr\_mvto(cr\_fb), cr_1) \\
\rightarrow & cr_2 \neq cr_{err} \wedge cr\_apos(cr_2) = cr\_aup \wedge \\
& cr\_pos(cr_2) = cr\_tofb \wedge (cr\_aloaded(cr_2) \leftrightarrow cr\_aloaded(cr_1)),
\end{aligned}$$

$$\begin{aligned}
& cr_1 = cr_{err} \vee cr\_apos(cr_1) \neq cr\_aup \vee cr\_pos(cr_1) \neq cr\_db \\
\rightarrow & cr\_trans(cr\_mvto(cr\_fb), cr_1) = cr_{err},
\end{aligned}$$

$$\begin{aligned}
& cr_1 \neq cr_{err} \wedge cr\_apos(cr_1) = cr\_aup \wedge cr\_pos(cr_1) = cr\_fb \wedge \\
& cr_2 = cr\_trans(cr\_mvto(cr\_db), cr_1) \\
\rightarrow & cr_2 \neq cr_{err} \wedge cr\_apos(cr_2) = cr\_aup \wedge cr\_pos(cr_2) = cr\_todb \wedge \\
& (cr\_aloaded(cr_2) \leftrightarrow cr\_aloaded(cr_1)),
\end{aligned}$$

$$\begin{aligned}
& cr_1 = cr_{err} \vee cr\_apos(cr_1) \neq cr\_aup \vee cr\_pos(cr_1) \neq cr\_fb \\
\rightarrow & cr\_trans(cr\_mvto(cr\_db), cr_1) = cr_{err},
\end{aligned}$$

$$\begin{aligned}
& cr_1 \neq cr_{err} \wedge cr\_apos(cr_1) = cr\_aup \wedge \\
& cr\_pos(cr_1) = cr\_tofb \wedge cr_2 = cr\_trans(cr\_at(cr\_fb), cr_1) \\
\rightarrow & cr_2 \neq cr_{err} \wedge cr\_apos(cr_2) = cr\_aup \wedge \\
& cr\_pos(cr_2) = cr\_fb \wedge (cr\_aloaded(cr_2) \leftrightarrow cr\_aloaded(cr_1)),
\end{aligned}$$

$$\begin{aligned}
& cr_1 = cr_{err} \vee cr\_apos(cr_1) \neq cr\_aup \vee cr\_pos(cr_1) \neq cr\_tofb \\
\rightarrow & cr\_trans(cr\_at(cr\_fb), cr_1) = cr_{err},
\end{aligned}$$

$$\begin{aligned}
& cr_1 \neq cr_{err} \wedge cr\_apos(cr_1) = cr\_aup \wedge \\
& cr\_pos(cr_1) = cr\_todb \wedge cr_2 = cr\_trans(cr\_at(cr\_db), cr_1) \\
\rightarrow & cr_2 \neq cr_{err} \wedge cr\_apos(cr_2) = cr\_aup \wedge \\
& cr\_pos(cr_2) = cr\_db \wedge (cr\_aloaded(cr_2) \leftrightarrow cr\_aloaded(cr_1)), \\
\\
& cr_1 = cr_{err} \vee cr\_apos(cr_1) \neq cr\_aup \vee cr\_pos(cr_1) \neq cr\_todb \\
\rightarrow & cr\_trans(cr\_at(cr\_fb), cr_1) = cr_{err}, \\
\\
& cr_1 \neq cr_{err} \wedge (cr\_apos(cr_1) = cr\_adown \vee cr\_apos(cr_1) = cr\_amid) \wedge \\
& (cr\_pos(cr_1) = cr\_fb \vee cr\_pos(cr_1) = cr\_db) \wedge cr_2 = cr\_trans(cr\_amvto(cr\_aup), cr_1) \\
\rightarrow & cr_2 \neq cr_{err} \wedge cr\_pos(cr_2) = cr\_pos(cr_1) \wedge \\
& cr\_apos(cr_2) = cr\_atoup \wedge (cr\_aloaded(cr_2) \leftrightarrow cr\_aloaded(cr_1)), \\
\\
& cr_1 = cr_{err} \vee cr\_apos(cr_1) \neq cr\_adown \wedge cr\_apos(cr_1) \neq cr\_amid \vee \\
& cr\_pos(cr_1) \neq cr\_fb \wedge cr\_pos(cr_1) \neq cr\_db \\
\rightarrow & cr\_trans(cr\_amvto(cr\_aup), cr_1) = cr_{err}, \\
\\
& cr_1 \neq cr_{err} \wedge (cr\_apos(cr_1) = cr\_adown \vee cr\_apos(cr_1) = cr\_aup) \wedge \\
& (cr\_pos(cr_1) = cr\_fb \vee cr\_pos(cr_1) = cr\_db) \wedge cr_2 = cr\_trans(cr\_amvto(cr\_amid), cr_1) \\
\rightarrow & cr_2 \neq cr_{err} \wedge cr\_pos(cr_2) = cr\_pos(cr_1) \wedge \\
& cr\_apos(cr_2) = cr\_atomid \wedge (cr\_aloaded(cr_2) \leftrightarrow cr\_aloaded(cr_1)), \\
\\
& cr_1 = cr_{err} \vee cr\_apos(cr_1) \neq cr\_adown \wedge cr\_apos(cr_1) \neq cr\_aup \vee \\
& cr\_pos(cr_1) \neq cr\_fb \wedge cr\_pos(cr_1) \neq cr\_db \\
\rightarrow & cr\_trans(cr\_amvto(cr\_amid), cr_1) = cr_{err}, \\
\\
& cr_1 \neq cr_{err} \wedge (cr\_apos(cr_1) = cr\_amid \vee cr\_apos(cr_1) = cr\_aup) \wedge \\
& (cr\_pos(cr_1) = cr\_fb \vee cr\_pos(cr_1) = cr\_db) \wedge cr_2 = cr\_trans(cr\_amvto(cr\_adown), cr_1) \\
\rightarrow & cr_2 \neq cr_{err} \wedge cr\_pos(cr_2) = cr\_pos(cr_1) \wedge \\
& cr\_apos(cr_2) = cr\_atodown \wedge (cr\_aloaded(cr_2) \leftrightarrow cr\_aloaded(cr_1)), \\
\\
& cr_1 = cr_{err} \vee cr\_apos(cr_1) \neq cr\_amid \wedge \\
& cr\_apos(cr_1) \neq cr\_aup \vee cr\_pos(cr_1) \neq cr\_fb \wedge cr\_pos(cr_1) \neq cr\_db \\
\rightarrow & cr\_trans(cr\_amvto(cr\_adown), cr_1) = cr_{err}, \\
\\
& cr_1 \neq cr_{err} \wedge cr\_apos(cr_1) = cr\_atoup \wedge \\
& (cr\_pos(cr_1) = cr\_fb \vee cr\_pos(cr_1) = cr\_db) \wedge cr_2 = cr\_trans(cr\_aat(cr\_aup), cr_1) \\
\rightarrow & cr_2 \neq cr_{err} \wedge cr\_pos(cr_2) = cr\_pos(cr_1) \wedge \\
& cr\_apos(cr_2) = cr\_aup \wedge (cr\_aloaded(cr_2) \leftrightarrow cr\_aloaded(cr_1)), \\
\\
& cr_1 = cr_{err} \vee cr\_apos(cr_1) \neq cr\_atoup \vee cr\_pos(cr_1) \neq cr\_fb \wedge cr\_pos(cr_1) \neq cr\_db \\
\rightarrow & cr\_trans(cr\_aat(cr\_aup), cr_1) = cr_{err}, \\
\\
& cr_1 \neq cr_{err} \wedge cr\_apos(cr_1) = cr\_atomid \wedge \\
& (cr\_pos(cr_1) = cr\_fb \vee cr\_pos(cr_1) = cr\_db) \wedge cr_2 = cr\_trans(cr\_aat(cr\_amid), cr_1) \\
\rightarrow & cr_2 \neq cr_{err} \wedge cr\_pos(cr_2) = cr\_pos(cr_1) \wedge \\
& cr\_apos(cr_2) = cr\_amid \wedge (cr\_aloaded(cr_2) \leftrightarrow cr\_aloaded(cr_1)), \\
\\
& cr_1 = cr_{err} \vee cr\_apos(cr_1) \neq cr\_atomid \vee cr\_pos(cr_1) \neq cr\_fb \wedge cr\_pos(cr_1) \neq cr\_db \\
\rightarrow & cr\_trans(cr\_aat(cr\_amid), cr_1) = cr_{err},
\end{aligned}$$



$$\begin{aligned}
& cr_1 \neq cr_{err} \wedge cr\_apos(cr_1) = cr\_atodown \wedge \\
& (cr\_pos(cr_1) = cr\_fb \vee cr\_pos(cr_1) = cr\_db) \wedge cr_2 = cr\_trans(cr\_aat(cr\_adown), cr_1) \\
\rightarrow & cr_2 \neq cr_{err} \wedge cr\_pos(cr_2) = cr\_pos(cr_1) \wedge \\
& cr\_apos(cr_2) = cr\_adown \wedge (cr\_aloaded(cr_2) \leftrightarrow cr\_aloaded(cr_1)), \\
\\
& cr_1 = cr_{err} \vee cr\_apos(cr_1) \neq cr\_atodown \vee cr\_pos(cr_1) \neq cr\_fb \wedge cr\_pos(cr_1) \neq cr\_db \\
\rightarrow & cr\_trans(cr\_aat(cr\_adown), cr_1) = cr_{err}, \\
\\
& cr_1 \neq cr_{err} \wedge (cr\_apos(cr_1) = cr\_adown \vee cr\_apos(cr_1) = cr\_amid) \wedge \\
& \neg cr\_aloaded(cr_1) \wedge cr_2 = cr\_trans(cr\_load, cr_1) \\
\rightarrow & cr_2 \neq cr_{err} \wedge cr\_aloaded(cr_2) \wedge \\
& cr\_pos(cr_2) = cr\_pos(cr_1) \wedge cr\_apos(cr_2) = cr\_apos(cr_1), \\
\\
& cr_1 = cr_{err} \vee cr\_apos(cr_1) \neq cr\_adown \wedge \\
& cr\_apos(cr_1) \neq cr\_amid \vee cr\_aloaded(cr_1) \\
\rightarrow & cr\_trans(cr\_load, cr_1) = cr_{err}, \\
\\
& cr_1 \neq cr_{err} \wedge (cr\_apos(cr_1) = cr\_adown \vee cr\_apos(cr_1) = cr\_amid) \wedge \\
& cr\_aloaded(cr_1) \wedge cr_2 = cr\_trans(cr\_unload, cr_1) \\
\rightarrow & cr_2 \neq cr_{err} \wedge \neg cr\_aloaded(cr_2) \wedge \\
& cr\_pos(cr_2) = cr\_pos(cr_1) \wedge cr\_apos(cr_2) = cr\_apos(cr_1), \\
\\
& cr_1 = cr_{err} \vee cr\_apos(cr_1) \neq cr\_adown \wedge \\
& cr\_apos(cr_1) \neq cr\_amid \vee \neg cr\_aloaded(cr_1) \\
\rightarrow & cr\_trans(cr\_unload, cr_1) = cr_{err}
\end{aligned}$$

**end enrich**

### A.3 Specification of Devices in the Production Cell Context

FBELT =  
**enrich** GENERIC\_FBELT **with**  
**functions** fb\_tr : fb\_event  $\times$  fb\_state  $\rightarrow$  fb\_state ;  
**axioms**  
fb\_e1  $\neq$  fb\_unload  $\rightarrow$  fb\_tr(fb\_e1, fb\_1) = fb\_trans(fb\_e1, fb\_1),  
fb\_tr(fb\_unload, fb\_1) = fb\_err

**end enrich**

TABLE =  
**enrich** GENERIC\_TABLE **with**  
**functions** ta\_tr : ta\_event  $\times$  ta\_state  $\rightarrow$  ta\_state ;  
**axioms**  
hte\_1  $\neq$  ta\_load  $\wedge$  hte\_1  $\neq$  ta\_unload  $\rightarrow$  ta\_tr(hte\_1, ta\_1) = ta\_trans(hte\_1, ta\_1),  
ta\_pos(ta\_1)  $\neq$  ta\_down  $\rightarrow$  ta\_tr(ta\_load, ta\_1) = ta\_err,  
ta\_pos(ta\_1) = ta\_down  $\rightarrow$  ta\_tr(ta\_load, ta\_1) = ta\_trans(ta\_load, ta\_1),  
ta\_tpos(ta\_1)  $\neq$  ta\_ro  $\rightarrow$  ta\_tr(ta\_unload, ta\_1) = ta\_err,  
ta\_tpos(ta\_1) = ta\_ro  $\rightarrow$  ta\_tr(ta\_unload, ta\_1) = ta\_trans(ta\_unload, ta\_1)

**end enrich**

ROBOT =

**enrich** GENERIC\_ROBOT **with**

**functions** ro\_tr : ro\_event  $\times$  ro\_state  $\rightarrow$  ro\_state ;

**axioms**

ro\_tr(ro\_at(ro\_po<sub>1</sub>), ro<sub>1</sub>) = ro\_trans(ro\_at(ro\_po<sub>1</sub>), ro<sub>1</sub>),  
ro\_tr(ro\_mvto(ro\_po<sub>1</sub>), ro<sub>1</sub>) = ro\_trans(ro\_mvto(ro\_po<sub>1</sub>), ro<sub>1</sub>),

ro\_pos(ro<sub>1</sub>) = ro\_alta  $\wedge$  ro\_alpos(ro<sub>1</sub>) = ro\_amid  
 $\rightarrow$  ro\_tr(ro\_alload, ro<sub>1</sub>) = ro\_trans(ro\_alload, ro<sub>1</sub>),

ro\_pos(ro<sub>1</sub>)  $\neq$  ro\_alta  $\vee$  ro\_alpos(ro<sub>1</sub>)  $\neq$  ro\_amid  
 $\rightarrow$  ro\_tr(ro\_alload, ro<sub>1</sub>) = roerr,

ro\_pos(ro<sub>1</sub>) = ro\_alpr  $\wedge$  ro\_alpos(ro<sub>1</sub>) = ro\_aout  
 $\rightarrow$  ro\_tr(ro\_alunload, ro<sub>1</sub>) = ro\_trans(ro\_alunload, ro<sub>1</sub>),

ro\_pos(ro<sub>1</sub>)  $\neq$  ro\_alpr  $\vee$  ro\_alpos(ro<sub>1</sub>)  $\neq$  ro\_aout  
 $\rightarrow$  ro\_tr(ro\_alunload, ro<sub>1</sub>) = ro\_trans(ro\_alunload, ro<sub>1</sub>),

ro\_pos(ro<sub>1</sub>) = ro\_a2pr  $\wedge$  ro\_a2pos(ro<sub>1</sub>) = ro\_aout  
 $\rightarrow$  ro\_tr(ro\_a2load, ro<sub>1</sub>) = ro\_trans(ro\_a2load, ro<sub>1</sub>),

ro\_pos(ro<sub>1</sub>)  $\neq$  ro\_a2pr  $\vee$  ro\_a2pos(ro<sub>1</sub>)  $\neq$  ro\_aout  
 $\rightarrow$  ro\_tr(ro\_a2load, ro<sub>1</sub>) = roerr,

ro\_pos(ro<sub>1</sub>) = ro\_a2db  $\wedge$  ro\_a2pos(ro<sub>1</sub>) = ro\_amid  
 $\rightarrow$  ro\_tr(ro\_a2unload, ro<sub>1</sub>) = ro\_trans(ro\_a2load, ro<sub>1</sub>),

ro\_pos(ro<sub>1</sub>)  $\neq$  ro\_a2db  $\vee$  ro\_a2pos(ro<sub>1</sub>)  $\neq$  ro\_amid  
 $\rightarrow$  ro\_tr(ro\_a2unload, ro<sub>1</sub>) = roerr,

ro\_tr(ro\_alat(roap<sub>1</sub>), ro<sub>1</sub>) = ro\_trans(ro\_alat(roap<sub>1</sub>), ro<sub>1</sub>),  
ro\_tr(ro\_almv(roap<sub>1</sub>), ro<sub>1</sub>) = ro\_trans(ro\_almv(roap<sub>1</sub>), ro<sub>1</sub>),  
ro\_tr(ro\_a2at(roap<sub>1</sub>), ro<sub>1</sub>) = ro\_trans(ro\_a2at(roap<sub>1</sub>), ro<sub>1</sub>),  
ro\_tr(ro\_a2mv(roap<sub>1</sub>), ro<sub>1</sub>) = ro\_trans(ro\_a2mv(roap<sub>1</sub>), ro<sub>1</sub>)

**end enrich**

PRESS =

**enrich** GENERIC\_PRESS **with**

**functions** pr\_tr : pr\_event  $\times$  pr\_state  $\rightarrow$  pr\_state ;

**axioms**

pr\_tr(pr\_at(prpo<sub>1</sub>), pr<sub>1</sub>) = pr\_trans(pr\_at(prpo<sub>1</sub>), pr<sub>1</sub>),  
pr\_tr(pr\_mvto(prpo<sub>1</sub>), pr<sub>1</sub>) = pr\_trans(pr\_mvto(prpo<sub>1</sub>), pr<sub>1</sub>),  
pr\_pos(pr<sub>1</sub>) = pr\_mid  $\rightarrow$  pr\_tr(pr\_load, pr<sub>1</sub>) = pr\_trans(pr\_load, pr<sub>1</sub>),  
pr\_pos(pr<sub>1</sub>)  $\neq$  pr\_mid  $\rightarrow$  pr\_tr(pr\_load, pr<sub>1</sub>) = prerr,  
pr\_pos(pr<sub>1</sub>) = pr\_down  $\rightarrow$  pr\_tr(pr\_unload, pr<sub>1</sub>) = pr\_trans(pr\_unload, pr<sub>1</sub>),  
pr\_pos(pr<sub>1</sub>)  $\neq$  pr\_down  $\rightarrow$  pr\_tr(pr\_unload, pr<sub>1</sub>) = prerr

**end enrich**

```

DBELT =
enrich GENERIC_DBELT with
  functions db_tr : db_event  $\times$  db_state  $\rightarrow$  db_state ;
  axioms
    dbe1  $\neq$  db_sheetfalls  $\rightarrow$  db_tr(dbe1, db1) = db_trans(dbe1, db1),
    db_tr(db_sheetfalls, db1) = dberr
end enrich

```

```

CRANE =
enrich GENERIC_CRANE with
  functions cr_tr : cr_event  $\times$  cr_state  $\rightarrow$  cr_state ;
  axioms
    cr_tr(cr_mvto(crpo1), cr1) = cr_trans(cr_mvto(crpo1), cr1),

    crap1  $\neq$  cr_adown  $\vee$  cr_pos(cr1)  $\neq$  cr_fb
     $\rightarrow$  cr_tr(cr_amvto(crap1), cr1) = cr_trans(cr_amvto(crap1), cr1),

    cr_pos(cr1) = cr_fb
     $\rightarrow$  cr_tr(cr_amvto(cr_adown), cr1) = crerr,

    crap1  $\neq$  cr_adown  $\vee$  cr_pos(cr1)  $\neq$  cr_fb
     $\rightarrow$  cr_tr(cr_aat(crap1), cr1) = cr_trans(cr_aat(crap1), cr1),

    cr_pos(cr1) = cr_fb  $\rightarrow$  cr_tr(cr_aat(cr_adown), cr1) = crerr,
    cr_pos(cr1) = cr_db  $\wedge$  cr_apos(cr1) = cr_adown
     $\rightarrow$  cr_tr(cr_load, cr1) = cr_trans(cr_load, cr1),

    cr_pos(cr1)  $\neq$  cr_db  $\vee$  cr_apos(cr1)  $\neq$  cr_adown
     $\rightarrow$  cr_tr(cr_load, cr1) = crerr,

    cr_pos(cr1) = cr_fb  $\wedge$  cr_apos(cr1) = cr_amid
     $\rightarrow$  cr_tr(cr_unload, cr1) = cr_trans(cr_unload, cr1),

    cr_pos(cr1)  $\neq$  cr_fb  $\vee$  cr_apos(cr1)  $\neq$  cr_amid
     $\rightarrow$  cr_tr(cr_unload, cr1) = crerr
end enrich

```

## A.4 The System Specification

```

ELEM =
specification
  sorts elem;
  variables a: elem;
end specification

```

LIST =

**generic specification**

**parameter** ELEM target

**sorts** list;

**constants** nil : list;

**functions**

car : list → elem ;  
cons : elem × list → list ;  
cdr : list → list ;  
append : list × list → list ;  
reverse : list → list ;

**variables** l<sub>1</sub>, l: list;

**axioms**

list **generated by** nil, cons;  
car(cons(a, l)) = a,  
cdr(cons(a, l)) = l,  
nil ≠ cons(a, l),  
l = nil ∨ l = cons(car(l), cdr(l)),  
append(nil, l) = l,  
append(cons(a, l<sub>1</sub>), l) = cons(a, append(l<sub>1</sub>, l)),  
reverse(nil) = nil,  
reverse(cons(a, l)) = append(reverse(l), cons(a, nil))

**end generic specification**

PROD\_CELL =

**enrich** FBELT + TABLE + ROBOT + PRESS + DBELT + CRANE **with**

**sorts** state, event;

**constants** init : state;

**functions**

mkstate : fb\_state × ta\_state × ro\_state ×  
pr\_state × db\_state × cr\_state → state ;  
trans : event × state → state ;  
fbc : state → fb\_state ;  
tac : state → ta\_state ;  
roc : state → ro\_state ;  
prc : state → pr\_state ;  
dbc : state → db\_state ;  
crc : state → cr\_state ;  
cfb : fb\_ev → event ;  
cht : ta\_ev → event ;  
cro : ro\_ev → event ;  
cpr : pr\_ev → event ;  
cdb : db\_ev → event ;  
ccr : cr\_ev → event ;

**predicates**

safe : state;  
notshared : event;  
expe : event × state;

**variables** e<sub>0</sub>, e: event; fz<sub>2</sub>, fz<sub>1</sub>, s<sub>2</sub>, s<sub>1</sub>, s: state;

fb<sub>e2</sub>: fb\_ev; ht<sub>e2</sub>: ta\_ev; ro<sub>e2</sub>: ro\_ev; cr<sub>e2</sub>: cr\_ev; pr<sub>e2</sub>: pr\_ev; db<sub>e2</sub>: db\_ev;

## axioms

state **generated by** mkstate;  
 event **generated by** cfb, cht, cro, cpr, cdb, ccr;

init = mkstate(fbinit, htinit, roinit, prinit, dbinit, crinit),

safe(mkstate(fb<sub>1</sub>, ta<sub>1</sub>, ro<sub>1</sub>, pr<sub>1</sub>, db<sub>1</sub>, cr<sub>1</sub>))  
 $\leftrightarrow$  fb<sub>1</sub>  $\neq$  fberr  $\wedge$  ta<sub>1</sub>  $\neq$  taerr  $\wedge$  ro<sub>1</sub>  $\neq$  roerr  $\wedge$   
 pr<sub>1</sub>  $\neq$  prerr  $\wedge$  db<sub>1</sub>  $\neq$  dberr  $\wedge$  cr<sub>1</sub>  $\neq$  cerr  $\wedge$   
 $\neg$  (ro\_pos(ro<sub>1</sub>) = ro\_a1pr  $\wedge$  ro\_a1pos(ro<sub>1</sub>)  $\neq$  ro\_ain)  $\wedge$   
 $\neg$  (ro\_pos(ro<sub>1</sub>) = ro\_a2pr  $\wedge$  ro\_a2pos(ro<sub>1</sub>)  $\neq$  ro\_ain),

fb(mkstate(fb<sub>1</sub>, ta<sub>1</sub>, ro<sub>1</sub>, pr<sub>1</sub>, db<sub>1</sub>, cr<sub>1</sub>)) = fb<sub>1</sub>,  
 tac(mkstate(fb<sub>1</sub>, ta<sub>1</sub>, ro<sub>1</sub>, pr<sub>1</sub>, db<sub>1</sub>, cr<sub>1</sub>)) = ta<sub>1</sub>,  
 roc(mkstate(fb<sub>1</sub>, ta<sub>1</sub>, ro<sub>1</sub>, pr<sub>1</sub>, db<sub>1</sub>, cr<sub>1</sub>)) = ro<sub>1</sub>,  
 prc(mkstate(fb<sub>1</sub>, ta<sub>1</sub>, ro<sub>1</sub>, pr<sub>1</sub>, db<sub>1</sub>, cr<sub>1</sub>)) = pr<sub>1</sub>,  
 dbc(mkstate(fb<sub>1</sub>, ta<sub>1</sub>, ro<sub>1</sub>, pr<sub>1</sub>, db<sub>1</sub>, cr<sub>1</sub>)) = db<sub>1</sub>,  
 crc(mkstate(fb<sub>1</sub>, ta<sub>1</sub>, ro<sub>1</sub>, pr<sub>1</sub>, db<sub>1</sub>, cr<sub>1</sub>)) = cr<sub>1</sub>,

s<sub>1</sub> = s<sub>2</sub>  
 $\leftrightarrow$  fbc(s<sub>1</sub>) = fbc(s<sub>2</sub>)  $\wedge$  tac(s<sub>1</sub>) = tac(s<sub>2</sub>)  $\wedge$  roc(s<sub>1</sub>) = roc(s<sub>2</sub>)  $\wedge$   
 prc(s<sub>1</sub>) = prc(s<sub>2</sub>)  $\wedge$  dbc(s<sub>1</sub>) = dbc(s<sub>2</sub>)  $\wedge$  crc(s<sub>1</sub>) = crc(s<sub>2</sub>),

$\forall e. (\exists fbe_1.e = cfb(fbe_1)) \vee (\exists hte_1.e = cht(hte_1)) \vee (\exists roe_1.e = cro(roe_1)) \vee$   
 $(\exists pre_1.e = cpr(pre_1)) \vee (\exists dbe_1.e = cdb(dbe_1)) \vee (\exists cre_1.e = ccr(cre_1)),$

cfb(fbe<sub>1</sub>) = cfb(fbe<sub>2</sub>)  $\leftrightarrow$  fbe<sub>1</sub> = fbe<sub>2</sub>,  
 cht(hte<sub>1</sub>) = cht(hte<sub>2</sub>)  $\leftrightarrow$  hte<sub>1</sub> = hte<sub>2</sub>,  
 cro(roe<sub>1</sub>) = cro(roe<sub>2</sub>)  $\leftrightarrow$  roe<sub>1</sub> = roe<sub>2</sub>,  
 cpr(pre<sub>1</sub>) = cpr(pre<sub>2</sub>)  $\leftrightarrow$  pre<sub>1</sub> = pre<sub>2</sub>,  
 cdb(dbe<sub>1</sub>) = cdb(dbe<sub>2</sub>)  $\leftrightarrow$  dbe<sub>1</sub> = dbe<sub>2</sub>,  
 ccr(cre<sub>1</sub>) = ccr(cre<sub>2</sub>)  $\leftrightarrow$  cre<sub>1</sub> = cre<sub>2</sub>,

cht(ta\_load) = cfb(fb\_sheetfalls),  
 cht(ta\_unload) = cro(ro\_a1load),  
 cro(ro\_a1unload) = cpr(pr\_load),  
 cpr(pr\_unload) = cro(ro\_a2load),  
 cro(ro\_a2unload) = cdb(db\_load),  
 cdb(db\_unload) = ccr(cr\_load),  
 ccr(cr\_unload) = cfb(fb\_load),

fbe<sub>1</sub>  $\neq$  fb\_unload  $\wedge$  fbe<sub>1</sub>  $\neq$  fb\_sheetfalls  $\rightarrow$  notshared(cfb(fbe<sub>1</sub>)),  
 hte<sub>1</sub>  $\neq$  ta\_load  $\wedge$  hte<sub>1</sub>  $\neq$  ta\_unload  $\rightarrow$  notshared(cht(hte<sub>1</sub>)),  
 roe<sub>1</sub>  $\neq$  ro\_a1load  $\wedge$  roe<sub>1</sub>  $\neq$  ro\_a1unload  $\wedge$  roe<sub>1</sub>  $\neq$  ro\_a2load  $\wedge$  roe<sub>1</sub>  $\neq$  ro\_a2unload  
 $\rightarrow$  notshared(cro(roe<sub>1</sub>)),  
 pre<sub>1</sub>  $\neq$  pr\_load  $\wedge$  pre<sub>1</sub>  $\neq$  pr\_unload  $\rightarrow$  notshared(cpr(pre<sub>1</sub>)),  
 dbe<sub>1</sub>  $\neq$  db\_load  $\wedge$  dbe<sub>1</sub>  $\neq$  db\_unload  $\rightarrow$  notshared(cdb(dbe<sub>1</sub>)),  
 cre<sub>1</sub>  $\neq$  cr\_load  $\wedge$  cre<sub>1</sub>  $\neq$  cr\_unload  $\rightarrow$  notshared(ccr(cre<sub>1</sub>)),

notshared(cfb(fbe<sub>1</sub>))  $\wedge$  notshared(cht(hte<sub>2</sub>))  $\rightarrow$  cfb(fbe<sub>1</sub>)  $\neq$  cht(hte<sub>2</sub>),  
 notshared(cfb(fbe<sub>1</sub>))  $\wedge$  notshared(cro(roe<sub>2</sub>))  $\rightarrow$  cfb(fbe<sub>1</sub>)  $\neq$  cro(roe<sub>2</sub>),  
 notshared(cfb(fbe<sub>1</sub>))  $\wedge$  notshared(cpr(pre<sub>2</sub>))  $\rightarrow$  cfb(fbe<sub>1</sub>)  $\neq$  cpr(pre<sub>2</sub>),  
 notshared(cfb(fbe<sub>1</sub>))  $\wedge$  notshared(cdb(dbe<sub>2</sub>))  $\rightarrow$  cfb(fbe<sub>1</sub>)  $\neq$  cdb(dbe<sub>2</sub>),

$\text{notshared}(\text{cfb}(\text{fbe}_1)) \wedge \text{notshared}(\text{ccr}(\text{cre}_2)) \rightarrow \text{cfb}(\text{fbe}_1) \neq \text{ccr}(\text{cre}_2),$   
 $\text{notshared}(\text{cht}(\text{hte}_1)) \wedge \text{notshared}(\text{cro}(\text{roe}_2)) \rightarrow \text{cht}(\text{hte}_1) \neq \text{cro}(\text{roe}_2),$   
 $\text{notshared}(\text{cht}(\text{hte}_1)) \wedge \text{notshared}(\text{cpr}(\text{pre}_2)) \rightarrow \text{cht}(\text{hte}_1) \neq \text{cpr}(\text{pre}_2),$   
 $\text{notshared}(\text{cht}(\text{hte}_1)) \wedge \text{notshared}(\text{cdb}(\text{dbe}_2)) \rightarrow \text{cht}(\text{hte}_1) \neq \text{cdb}(\text{dbe}_2),$   
 $\text{notshared}(\text{cht}(\text{hte}_1)) \wedge \text{notshared}(\text{ccr}(\text{cre}_2)) \rightarrow \text{cht}(\text{hte}_1) \neq \text{ccr}(\text{cre}_2),$   
 $\text{notshared}(\text{cro}(\text{roe}_1)) \wedge \text{notshared}(\text{cpr}(\text{pre}_2)) \rightarrow \text{cro}(\text{roe}_1) \neq \text{cpr}(\text{pre}_2),$   
 $\text{notshared}(\text{cro}(\text{roe}_1)) \wedge \text{notshared}(\text{cdb}(\text{dbe}_2)) \rightarrow \text{cro}(\text{roe}_1) \neq \text{cdb}(\text{dbe}_2),$   
 $\text{notshared}(\text{cro}(\text{roe}_1)) \wedge \text{notshared}(\text{ccr}(\text{cre}_2)) \rightarrow \text{cro}(\text{roe}_1) \neq \text{ccr}(\text{cre}_2),$   
 $\text{notshared}(\text{cpr}(\text{pre}_1)) \wedge \text{notshared}(\text{cdb}(\text{dbe}_2)) \rightarrow \text{cpr}(\text{pre}_1) \neq \text{cdb}(\text{dbe}_2),$   
 $\text{notshared}(\text{cpr}(\text{pre}_1)) \wedge \text{notshared}(\text{ccr}(\text{cre}_2)) \rightarrow \text{cpr}(\text{pre}_1) \neq \text{ccr}(\text{cre}_2),$   
 $\text{notshared}(\text{cpr}(\text{pre}_1)) \wedge \text{notshared}(\text{cdb}(\text{dbe}_2)) \rightarrow \text{cpr}(\text{pre}_1) \neq \text{ccr}(\text{cre}_2),$

$\text{notshared}(\text{cfb}(\text{fbe}_1))$   
 $\rightarrow \text{trans}(\text{cfb}(\text{fbe}_1), \text{fz}_1) =$   
 $\text{mkstate}(\text{fb\_tr}(\text{fbe}_1, \text{fbc}(\text{fz}_1)), \text{tac}(\text{fz}_1), \text{roc}(\text{fz}_1), \text{prc}(\text{fz}_1), \text{dbc}(\text{fz}_1), \text{crc}(\text{fz}_1)),$

$\text{notshared}(\text{cht}(\text{hte}_1))$   
 $\rightarrow \text{trans}(\text{cht}(\text{hte}_1), \text{fz}_1) =$   
 $\text{mkstate}(\text{fbc}(\text{fz}_1), \text{ta\_tr}(\text{hte}_1, \text{tac}(\text{fz}_1)), \text{roc}(\text{fz}_1), \text{prc}(\text{fz}_1), \text{dbc}(\text{fz}_1), \text{crc}(\text{fz}_1)),$

$\text{notshared}(\text{cro}(\text{roe}_1))$   
 $\rightarrow \text{trans}(\text{cro}(\text{roe}_1), \text{fz}_1) =$   
 $\text{mkstate}(\text{fbc}(\text{fz}_1), \text{tac}(\text{fz}_1), \text{ro\_tr}(\text{roe}_1, \text{roc}(\text{fz}_1)), \text{prc}(\text{fz}_1), \text{dbc}(\text{fz}_1), \text{crc}(\text{fz}_1)),$

$\text{notshared}(\text{cpr}(\text{pre}_1))$   
 $\rightarrow \text{trans}(\text{cpr}(\text{pre}_1), \text{fz}_1) =$   
 $\text{mkstate}(\text{fbc}(\text{fz}_1), \text{tac}(\text{fz}_1), \text{roc}(\text{fz}_1), \text{pr\_tr}(\text{pre}_1, \text{prc}(\text{fz}_1)), \text{dbc}(\text{fz}_1), \text{crc}(\text{fz}_1)),$

$\text{notshared}(\text{cdb}(\text{dbe}_1))$   
 $\rightarrow \text{trans}(\text{cdb}(\text{dbe}_1), \text{fz}_1) =$   
 $\text{mkstate}(\text{fbc}(\text{fz}_1), \text{tac}(\text{fz}_1), \text{roc}(\text{fz}_1), \text{prc}(\text{fz}_1), \text{db\_tr}(\text{dbe}_1, \text{dbc}(\text{fz}_1)), \text{crc}(\text{fz}_1)),$

$\text{notshared}(\text{ccr}(\text{cre}_1))$   
 $\rightarrow \text{trans}(\text{ccr}(\text{cre}_1), \text{fz}_1) =$   
 $\text{mkstate}(\text{fbc}(\text{fz}_1), \text{tac}(\text{fz}_1), \text{roc}(\text{fz}_1), \text{prc}(\text{fz}_1), \text{dbc}(\text{fz}_1), \text{cr\_tr}(\text{cre}_1, \text{crc}(\text{fz}_1))),$

$\text{trans}(\text{cfb}(\text{fb\_sheetfalls}), \text{fz}_1)$   
 $= \text{mkstate}(\text{fb\_tr}(\text{fb\_sheetfalls}, \text{fbc}(\text{fz}_1)), \text{ta\_tr}(\text{ta\_load}, \text{tac}(\text{fz}_1)),$   
 $\text{roc}(\text{fz}_1), \text{prc}(\text{fz}_1), \text{dbc}(\text{fz}_1), \text{crc}(\text{fz}_1)),$

$\text{trans}(\text{cht}(\text{ta\_unload}), \text{fz}_1)$   
 $= \text{mkstate}(\text{fbc}(\text{fz}_1), \text{ta\_tr}(\text{ta\_unload}, \text{tac}(\text{fz}_1)), \text{ro\_tr}(\text{ro\_alload}, \text{roc}(\text{fz}_1)),$   
 $\text{prc}(\text{fz}_1), \text{dbc}(\text{fz}_1), \text{crc}(\text{fz}_1)),$

$\text{trans}(\text{cro}(\text{ro\_alunload}), \text{fz}_1)$   
 $= \text{mkstate}(\text{fbc}(\text{fz}_1), \text{tac}(\text{fz}_1), \text{ro\_tr}(\text{ro\_alunload}, \text{roc}(\text{fz}_1)),$   
 $\text{pr\_tr}(\text{pr\_load}, \text{prc}(\text{fz}_1)), \text{dbc}(\text{fz}_1), \text{crc}(\text{fz}_1)),$

$\text{trans}(\text{cpr}(\text{pr\_unload}), \text{fz}_1)$   
 $= \text{mkstate}(\text{fbc}(\text{fz}_1), \text{tac}(\text{fz}_1), \text{ro\_tr}(\text{ro\_a2load}, \text{roc}(\text{fz}_1)),$   
 $\text{pr\_tr}(\text{pr\_unload}, \text{prc}(\text{fz}_1)), \text{dbc}(\text{fz}_1), \text{crc}(\text{fz}_1)),$

```

    trans(cro(ro_a2unload), fz1)
= mkstate(fbc(fz1), tac(fz1), ro_tr(ro_a2unload, roc(fz1)),
          prc(fz1), db_tr(db_load, dbc(fz1)), crc(fz1)),

    trans(cdb(db_unload), fz1)
= mkstate(fbc(fz1), tac(fz1), roc(fz1),
          prc(fz1), db_tr(db_unload, dbc(fz1)), cr_tr(cr_load, crc(fz1))),

    trans(ccr(cr_unload), fz1)
= mkstate(fb_tr(fb_load, fbc(fz1)), tac(fz1), roc(fz1),
          prc(fz1), dbc(fz1), cr_tr(cr_unload, crc(fz1))),

    expe(cfb(fbe1), fz1) ↔ fb_exp(fbe1, fbc(fz1)),
    hte1 ≠ ta_load → (expe(cht(hte1), fz1) ↔ ta_exp(hte1, tac(fz1))),
    expe(cro(roe1), fz1) ↔ ro_exp(roe1, roc(fz1)),
    expe(cpr(pre1), fz1) ↔ pr_exp(pre1, prc(fz1)),
    expe(cdb(dbe1), fz1) ↔ db_exp(dbe1, dbc(fz1)),
    expe(ccr(cre1), fz1) ↔ cr_exp(cre1, crc(fz1))
end enrich

PROD_CELL_WITH_EVENTLIST =
actualize LIST with PROD_CELL by morphism
    elem → event, list → elist, nil → enil, cons → econs, car → ecar, cdr → ecdr,
    append → eappend, reverse → ereverse, a → e, l → el, l1 → el1
end actualize

PRODUCTION_CELL =
enrich PROD_CELL_WITH_EVENTLIST with
    functions trans* : elist × state → state ;
    variables el1 : elist;
    axioms
        trans*(enil, fz1) = fz1,
        trans*(econs(e, el), fz1) = trans*(el, trans(e, fz1))
end enrich

```

## B The Control Procedure

```
control#(e, z1; var el, z2)
begin
z1 := trans(e, z1);
el := enil;

if e = cfb(fb_pbinterruption) then
  begin if ta_pos(tac(z1)) ≠ ta_down then el := econs(cfb(fb_off), el) end

else
if e = cfb(fb_sheetfalls) then
  begin
  el := econs(cta(ta_mvto(ta_up)), el);
  if cr_apos(crc(z1)) = cr_amid then
    begin
    el := econs(ccr(cr_unload), el);
    el := econs(ccr(cr_amvto(cr_aup)), el)
    end
  else
    el := econs(cfb(fb_off), el)
  end

else
if e = cta(ta_at(ta_down)) then
  begin if fb_pbinterrupted(fbc(z1)) then el := econs(cfb(fb_on), el) end

else
if e = cta(ta_at(ta_up)) then el := econs(cta(ta_tmvtto(ta_ro)), el) else

if e = cta(ta_tat(ta_fb)) then el := econs(cta(ta_mvto(ta_down)), el) else

if e = cta(ta_tat(ta_ro)) then
  begin
  if ro_alpos(roc(z1)) = ro_amid then
    begin
    el := econs(cro(ro_alload), el);
    el := econs(cro(ro_almv(ro_ain)), el);
    el := econs(cta(ta_tmvtto(ta_fb)), el)
    end
  end

else
if e = cro(ro_alat(ro_amid)) then
  begin
  if ta_tpos(tac(z1)) = ta_ro then
    begin
    el := econs(cro(ro_alload), el);
    el := econs(cro(ro_almv(ro_ain)), el);
    el := econs(cta(ta_tmvtto(ta_fb)), el)
    end
  end
end
```



```

else
if e = cro(ro_a1at(ro_ain)) then
  if ro_pos(roc(z1)) = ro_a1ta then
    if ¬ pr_loaded(prc(z1)) then el := econs(cro(ro_mvto(ro_a1pr)), el) else
      if ro_a2loaded(roc(z1)) then el := econs(cro(ro_mvto(ro_a2db)), el) else
        el := econs(cro(ro_mvto(ro_a2pr)), el)
    else
      begin
        el := econs(cpr(pr_mvto(pr_up)), el);
        if ro_a2loaded(roc(z1)) then
          if db_loaded(dbc(z1)) then el := econs(cro(ro_mvto(ro_a1ta)), el) else
            el := econs(cro(ro_mvto(ro_a2db)), el)
          else
            el := econs(cro(ro_mvto(ro_a2pr)), el)
          end
        end
      end
else
  if e = cro(ro_a2at(ro_ain)) then
    if ro_pos(roc(z1)) = ro_a2pr then
      begin
        el := econs(cpr(pr_mvto(pr_mid)), el);
        if ro_a1loaded(roc(z1)) then el := econs(cro(ro_mvto(ro_a1pr)), el) else
          if db_loaded(dbc(z1)) then el := econs(cro(ro_mvto(ro_a1ta)), el) else
            el := econs(cro(ro_mvto(ro_a2db)), el)
          end
        end
      end
    else
      if pr_loaded(prc(z1)) then el := econs(cro(ro_mvto(ro_a2pr)), el) else
        if ro_a1loaded(roc(z1)) then el := econs(cro(ro_mvto(ro_a1pr)), el) else
          el := econs(cro(ro_mvto(ro_a1ta)), el)
        end
      end
    else
      if e = cro(ro_at(ro_a1pr)) then
        begin if pr_pos(prc(z1)) = pr_mid then el := econs(cro(ro_a1mv(ro_aout)), el) end
      else
        if e = cro(ro_at(ro_a1ta)) then el := econs(cro(ro_a1mv(ro_amid)), el) else
          if e = cro(ro_at(ro_a2pr)) then
            begin if pr_pos(prc(z1)) = pr_down then el := econs(cro(ro_a2mv(ro_aout)), el) end
          else
            if e = cro(ro_at(ro_a2db)) then
              begin if ¬ db_loaded(dbc(z1)) then el := econs(cro(ro_a2mv(ro_amid)), el) end
            else
              if e = cro(ro_a1at(ro_aout)) then
                begin
                  el := econs(cro(ro_a1unload), el);
                  el := econs(cro(ro_a1mv(ro_ain)), el)
                end
              end
            end
          end
        end
      end
    end
  end

```

```

else
if e = cro(ro_a2at(ro_amid)) then
  begin
    el := econs(cro(ro_a2unload), el);
    el := econs(cro(ro_a2mv(ro_ain)), el);
    el := econs(cdb(db_on), el)
  end

else
if e = cro(ro_a2at(ro_aout)) then
  begin el := econs(cro(ro_a2load), el); el := econs(cro(ro_a2mv(ro_ain)), el) end

else
if e = cpr(pr_at(pr_up)) then el := econs(cpr(pr_mvto(pr_down)), el) else

if e = cpr(pr_at(pr_down)) then
  begin if ro_pos(roc(z1)) = ro_a2pr then el := econs(cro(ro_a2mv(ro_aout)), el) end

else
if e = cpr(pr_at(pr_mid)) then
  begin if ro_pos(roc(z1)) = ro_a1pr then el := econs(cro(ro_a1mv(ro_aout)), el) end

else
if e = cdb(db_pbinterruption) then
  begin
    el := econs(cdb(db_off), el);
    if cr_apos(roc(z1)) = cr_adown then
      begin
        el := econs(ccr(cr_load), el);
        el := econs(ccr(cr_amvto(cr_aup)), el);
        if ro_pos(roc(z1)) = ro_a2db then el := econs(cro(ro_a2mv(ro_amid)), el)
      end
    end
  end

else
if e = ccr(cr_aat(cr_adown)) then
  begin
    if db_pbinterrupted(dbc(z1)) then
      begin
        el := econs(cdb(db_unload), el);
        el := econs(ccr(cr_amvto(cr_aup)), el);
        if ro_pos(roc(z1)) = ro_a2db then el := econs(cro(ro_a2mv(ro_amid)), el)
      end
    end
  end

```

```

else
if e = ccr(cr_aat(cr_amid)) then
  begin
    if  $\neg$  fb_loaded(fbc(z1)) then
      begin
        el := econs(ccr(cr_unload), el);
        el := econs(ccr(cr_amvto(cr_aup)), el);
        el := econs(cfb(fb_on), el)
      end
    end
  end

else
if e = ccr(cr_aat(cr_aup)) then
  if cr_loaded(crc(z1)) then el := econs(ccr(cr_mvto(cr_fb)), el) else
    el := econs(ccr(cr_mvto(cr_db)), el)

else
if e = ccr(cr_at(cr_fb)) then el := econs(ccr(cr_amvto(cr_amid)), el) else

if e = ccr(cr_at(cr_db)) then el := econs(ccr(cr_mvto(cr_adown)), el);

el := ereverse(el);
z2 := trans*(el, z1)
end

```

## C The Invariant

```
safe(z1)

and (   fb_loaded(fbc(z1))
  ->   (   fb_pbinterrupted(fbc(z1))
    -> (   not fb_running(fbc(z1))
      -> not ta_pos(tac(z1)) = ta_down ))
  and (   fb_pbinterrupted(fbc(z1))
    -> (   fb_running(fbc(z1))
      -> ta_pos(tac(z1)) = ta_down ))
  and (   not fb_pbinterrupted(fbc(z1))
    -> fb_running(fbc(z1)) ))

and (   not fb_loaded(fbc(z1))
  -> not fb_pbinterrupted(fbc(z1)) and not fb_running(fbc(z1)) )

and (ta_loaded(tac(z1))
  -> (   not ta_pos(tac(z1)) = ta_down
    and not ta_pos(tac(z1)) = ta_todown
    and not ta_tpos(tac(z1)) = ta_tofb
    and (   ta_pos(tac(z1)) = ta_up
      -> not ta_tpos(tac(z1)) = ta_fb )) )

and (not ta_loaded(tac(z1))
  -> (   not ta_pos(tac(z1)) = ta_toup
    and not ta_tpos(tac(z1)) = ta_ro
    and not ta_tpos(tac(z1)) = ta_toro
    and (   ta_tpos(tac(z1)) = ta_fb
      -> not ta_pos(tac(z1)) = ta_up)))

and (ro_ailloaded(roc(z1))
  -> (ro_pos(roc(z1)) = ro_a1ta
  -> ro_a1pos(roc(z1)) = ro_atoin))

and (ro_ailloaded(roc(z1))
  -> (ro_pos(roc(z1)) = ro_a1pr
  ->   ro_a1pos(roc(z1)) = ro_ain
    or ro_a1pos(roc(z1)) = ro_atoout))

and (ro_ailloaded(roc(z1))
  -> not ro_pos(roc(z1)) = ro_a1tota)

and (not ro_ailloaded(roc(z1))
  -> (ro_pos(roc(z1)) = ro_a1pr
  -> ro_a1pos(roc(z1)) = ro_atoin))

and (not ro_ailloaded(roc(z1))
  -> (ro_pos(roc(z1)) = ro_a1ta
  ->   ro_a1pos(roc(z1)) = ro_amid
    or ro_a1pos(roc(z1)) = ro_atomid))
```

```

and (not ro_a1loaded(roc(z1))
     -> not ro_pos(roc(z1)) = ro_a1upr)

and (ro_a2loaded(roc(z1))
     -> (ro_pos(roc(z1)) = ro_a2pr
         -> ro_a2pos(roc(z1)) = ro_atoin))

and (ro_a2loaded(roc(z1))
     -> (ro_pos(roc(z1)) = ro_a2db
         -> ro_a2pos(roc(z1)) = ro_ain
            or ro_a2pos(roc(z1)) = ro_atomid))

and (ro_a2loaded(roc(z1))
     -> not ro_pos(roc(z1)) = ro_a2upr)

and (not ro_a2loaded(roc(z1))
     -> (ro_pos(roc(z1)) = ro_a2pr
         -> ro_a2pos(roc(z1)) = ro_ain
            or ro_a2pos(roc(z1)) = ro_atoout))

and (not ro_a2loaded(roc(z1))
     -> not ro_pos(roc(z1)) = ro_a2todb)

and (not ro_a2loaded(roc(z1))
     -> (ro_pos(roc(z1)) = ro_a2db
         -> ro_a2pos(roc(z1)) = ro_atoin))

and (pr_loaded(prc(z1))
     -> (pr_pressed(prc(z1))
         -> pr_pos(prc(z1)) = pr_todown
            or pr_pos(prc(z1)) = pr_down))

and (pr_loaded(prc(z1))
     -> (not pr_pressed(prc(z1))
         -> ( pr_pos(prc(z1)) = pr_mid
            or pr_pos(prc(z1)) = pr_toup)))

and (not pr_loaded(prc(z1))
     -> (not pr_pos(prc(z1)) = pr_down
         -> pr_pos(prc(z1)) = pr_tomid
            or pr_pos(prc(z1)) = pr_mid))

and (pr_loaded(prc(z1))
     -> (pr_pos(prc(z1)) = pr_mid
         -> ro_a1pos(roc(z1)) = ro_atoin
            and ro_pos(roc(z1)) = ro_a1pr ))

and (pr_loaded(prc(z1))
     -> (not pr_pos(prc(z1)) = pr_mid
         -> not ro_pos(roc(z1)) = ro_a1upr
            and not ro_pos(roc(z1)) = ro_a1pr ))

```

```

and (not pr_loaded(prc(z1))
    -> (pr_pos(prc(z1)) = pr_down
        -> ro_pos(roc(z1)) = ro_a2pr
            and ro_a2pos(roc(z1)) = ro_atoin ))

and (not pr_loaded(prc(z1))
    -> (not pr_pos(prc(z1)) = pr_down
        -> not ro_pos(roc(z1)) = ro_a2pr
            and not ro_pos(roc(z1)) = ro_a2upr))

and ( db_loaded(dbc(z1))
    -> ( db_pbinterrupted(dbc(z1))
        -> not db_running(dbc(z1)) )
        and ( not db_pbinterrupted(dbc(z1))
            -> db_running(dbc(z1)) ))

and ( not db_loaded(dbc(z1))
    -> not db_running(dbc(z1)) and not db_pbinterrupted(dbc(z1)) )

and (cr_loaded(crc(z1))
    -> ( not cr_pos(crc(z1)) = cr_todb
        and (cr_pos(crc(z1)) = cr_fb
            -> cr_apos(crc(z1)) = cr_atomid
                or cr_apos(crc(z1)) = cr_amid)
        and (cr_pos(crc(z1)) = cr_db
            -> (cr_apos(crc(z1)) = cr_atoup))))

and (not cr_loaded(crc(z1))
    -> ( not cr_pos(crc(z1)) = cr_tofb
        and (cr_pos(crc(z1)) = cr_db
            -> ( cr_apos(crc(z1)) = cr_adown
                or cr_apos(crc(z1)) = cr_atodown))
        and (cr_pos(crc(z1)) = cr_fb
            -> cr_apos(crc(z1)) = cr_atoup)))

and ( ro_a1loaded(roc(z1))
    -> ( not ro_a2loaded(roc(z1))
        or not pr_loaded(prc(z1))
        or not ta_loaded(tac(z1))
        or not cr_loaded(crc(z1))
        or not fb_loaded(fbc(z1))
        or not db_loaded(dbc(z1)) ))

and ( not ro_a1loaded(roc(z1))
    -> ( ro_a2loaded(roc(z1))
        or pr_loaded(prc(z1))
        or ta_loaded(tac(z1))
        or cr_loaded(crc(z1))
        or fb_loaded(fbc(z1))
        or db_loaded(dbc(z1)) ))

```

```

and ( not pr_loaded(prc(z1))
      -> ( ro_a1loaded(roc(z1))
          -> ( pr_pos(prc(z1)) = pr_mid
              -> not ( ro_pos(roc(z1)) = ro_a1pr
                      and ro_a1pos(roc(z1)) = ro_ain))) )

and ( not ro_a2loaded(roc(z1))
      -> (pr_loaded(prc(z1))
          -> (pr_pos(prc(z1)) = pr_down
              -> not (ro_pos(roc(z1)) = ro_a2pr
                      and ro_a2pos(roc(z1)) = ro_ain))) )

and ( not ro_a1loaded(roc(z1))
      -> ( ta_loaded(tac(z1))
          -> ( ta_pos(tac(z1)) = ta_up
              -> (ta_tpos(tac(z1)) = ta_ro
                  -> not ( ro_pos(roc(z1)) = ro_a1ta
                          and ro_a1pos(roc(z1)) = ro_amid)))) )

and ( not ta_loaded(tac(z1))
      -> ( fb_loaded(fbc(z1))
          -> (fb_pbinterrupted(fbc(z1))
              -> (not fb_running(fbc(z1))
                  -> not ( ta_pos(tac(z1)) = ta_down
                          and ta_tpos(tac(z1)) = ta_fb)))) )

and ( not fb_loaded(fbc(z1))
      -> ( cr_aloaded(crc(z1))
          -> not ( cr_pos(crc(z1)) = cr_fb
                  and cr_apos(crc(z1)) = cr_amid) ) )

and ( not cr_aloaded(crc(z1))
      -> ( db_loaded(dbc(z1))
          -> ( db_pbinterrupted(dbc(z1))
              -> (not db_running(dbc(z1))
                  -> not ( cr_pos(crc(z1)) = cr_db
                          and cr_apos(crc(z1)) = cr_adown)))) )

and ( not db_loaded(dbc(z1))
      -> ( ro_a2loaded(roc(z1))
          -> not ( ro_pos(roc(z1)) = ro_a2db
                  and ro_a2pos(roc(z1)) = ro_ain) ) )

and ( ro_a1loaded(roc(z1))
      -> ( not pr_loaded(prc(z1))
          -> ( ro_pos(roc(z1)) = ro_a2db
              -> not (db_loaded(dbc(z1)) and ro_a2loaded(roc(z1))) )))

and ( ro_a1loaded(roc(z1))
      -> ( not pr_loaded(prc(z1))
          -> ( ro_pos(roc(z1)) = ro_a2todb
              -> not db_loaded(dbc(z1)) and ro_a2loaded(roc(z1)) )))

```

```

and (  not ro_a2loaded(roc(z1))
      -> (  pr_loaded(prc(z1))
          -> (  ro_pos(roc(z1)) = ro_a1ta
              -> not (not ta_loaded(tac(z1)) and not ro_a1loaded(roc(z1))) )))

and (  not ro_a2loaded(roc(z1))
      -> (  pr_loaded(prc(z1))
          -> (  ro_pos(roc(z1)) = ro_a1tota
              -> ta_loaded(tac(z1)) and not ro_a1loaded(roc(z1)) )))

and (  (  ro_a2loaded(roc(z1))
        -> (  not ro_a1loaded(roc(z1))
            -> (  db_loaded(dbc(z1))
                ->      not ro_pos(roc(z1)) = ro_a2db
                    and not ro_pos(roc(z1)) = ro_a2todb)))

and (  (  ro_a2loaded(roc(z1))
        -> (  not ro_a1loaded(roc(z1))
            -> (  not db_loaded(dbc(z1))
                -> (not cr_aloaded(crc(z1))
                    -> (not fb_loaded(fbc(z1))
                        -> (not ta_loaded(tac(z1))
                            ->      not ro_pos(roc(z1)) = ro_a1ta
                                and not ro_pos(roc(z1)) = ro_a1tota))))))

```



## References

- [Ha 79] D. Harel : *First Order Dynamic Logic*. Springer LNCS 1979.
- [HRS 89] M. Heisel, W. Reif, W. Stephan : *A Dynamic Logic for Program Verification*. “Logic at Botik” 89, Meyer, Taitslin (eds.), Springer LNCS 1989.
- [HRS 90] M. Heisel, W. Reif, W. Stephan : *Tactical Theorem Proving in Program Verification*. 10th International Conference on Automated Deduction, Kaiserslautern, FRG, Springer LNCS 1990.
- [Li 93] T. Lindner. Task Description for the Case Study “Production Cell”, Technical Report, Forschungszentrum Informatik, Haid-und-Neu-Stra”se 10-14, D-76131 Karlsruhe, 1993.
- [LL 94] C. Lewerentz, T. Lindner (eds.). Case Study “Production Cell”: A Comparative Study in Formal Specification and Verification. FZI Publication 1/94. Forschungszentrum Informatik, 1994.
- [Re 92] W. Reif : *Verification of Large Software Systems*. Conference on Foundations of Software Technology and Theoretical Computer Science, New Dehli, India, Shyamasundar (ed.), Springer LNCS 1992.