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 lifeness and safety conditions, is carried out using the KIV system.

Contents

1 Introduction 2

2 The Intended Model 2

3 Algebraic Specification of State Oriented Systems 2
- 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

4 The Control Program and its Correctness 9

5 Verification 11

6 Conclusions and Further Work 13

A The Specifications 14
- 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 The Control Procedure 40

C The Invariant 44

*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 lifeness 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 lifeness 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.

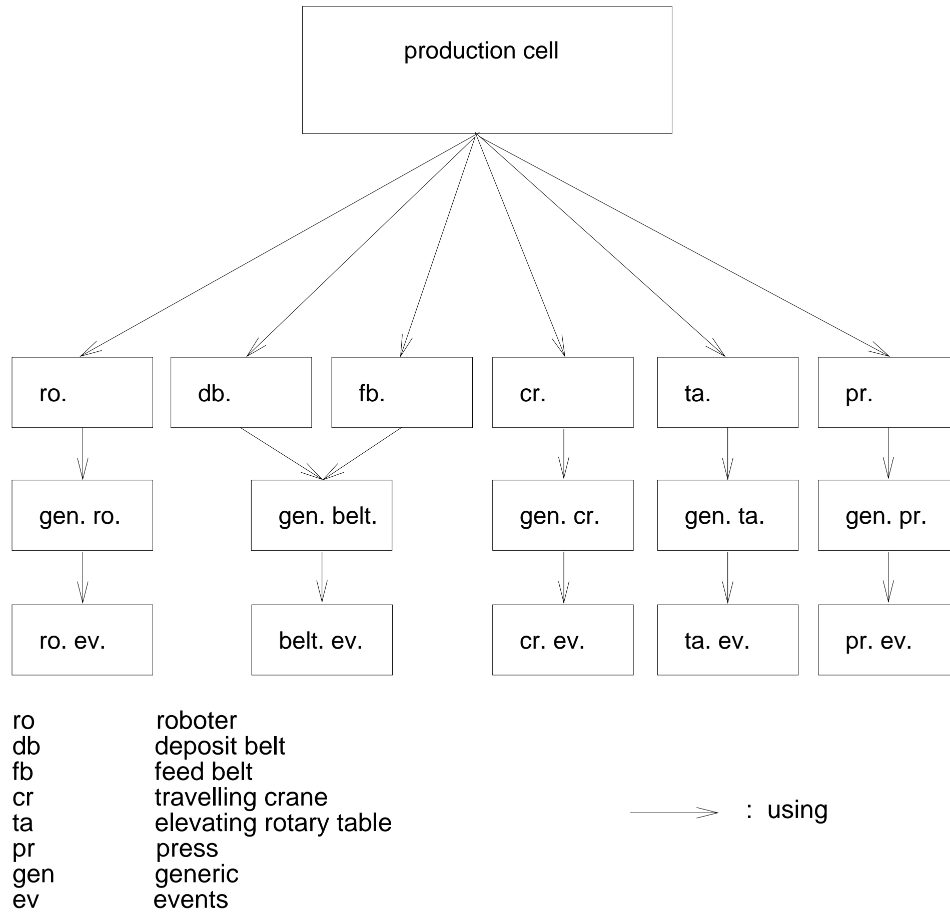

Fig.1: global specification

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_fb) 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 → ta_hposition;
             ta_vpos : ta_state → ta_vposition;
             ta_trans : ta_event, ta_state → 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 ≠ taerr
→ (     tas1 = tas2
     ↔    tas2≠ taerr
        ∧ ta_hpos(tas1) = ta_hpos(tas2) ∧ ta_vpos(tas1) = ta_vpos(tas2)
        ∧ (ta_hmoves(tas1) ↔ ta_hmoves(tas2))
        ∧ (ta_vmoves(tas1) ↔ ta_vmoves(tas2))
        ∧ (ta_loaded(tas1) ↔ ta_loaded(tas2)) ),

ta_vpos(tas) ≠ ta_up ∧ ta_hmoves(tas) → tas = taerr,

ta_exp(ta_hat(ta_fb),tas) ↔ tas ≠ taerr ∧ ta_hmoves(tas) ∧ ta_hpos(tas) = ta_fb,
....
¬ ta_exp(ta_load),
....
   ta_trans(ta_hto(ta_fb),tas) ≠ taerr
↔    tas ≠ taerr ∧ ta_hpos(tas) = ta_ro ∧ ta_vpos(tas) = ta_up
   ∧ ¬ ta_hmoves(tas) ∧ ¬ ta_vmoves(tas),

   tas1 = ta_trans(ta_hto(ta_fb),tas) ∧ tas1 ≠ taerr
→    ta_hpos(tas1) = ta_fb ∧ ta_vpos(tas1) = ta_up ∧ ta_hmoves(tas1)
   ∧ ¬ ta_vmoves(tas1) ∧ (ta_loaded(tas1) ↔ ta_loaded(tas)),

   ta_trans(ta_load,tas) ≠ taerr
↔ tas ≠ taerr ∧ ¬ ta_hmoves(tas) ∧ ¬ ta_vmoves(tas) ∧ ¬ ta_loaded(tas)
```

$tas1 = ta_trans(ta_load,tas) \land tas1 \neq taerr$
$\rightarrow \quad ta_hpos(tas1) = ta_hpos(tas) \land ta_vpos(tas1) = ta_vpos(tas)$
$\land \neg\, ta_hmoves(tas) \land \neg\, ta_vmoves(tas) \land ta_loaded(tas),$

....

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 crashs 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 → ta_state;

  axioms    ¬(tae = ta_load ∨ tae = ta_unload)
          → ta_tr(tae,tas) = ta_trans(tae,tas),

            ¬(ta_hpos(tas) = ta_fb ∧ ta_vpos(tas) = ta_down)
          → ta_tr(ta_load,tas) = taerr,

            ta_hpos(tas) = ta_fb ∧ ta_vpos(tas) = ta_down
          → ta_tr(ta_load,tas) = ta_trans(ta_load,tas),

            ¬(ta_hpos(tas) = ta_ro ∧ ta_vpos(tas) = ta_up)
          → ta_tr(ta_unload,tas) = taerr,

            ta_hpos(tas) = ta_ro ∧ ta_vpos(tas) = ta_up
          → 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 → event;
                    cta : ta_event → event;
                    cro : ro_event → event;
                    cpr : pr_event → event;
                    cdb : db_event → event;
                    ccr : cr_event → event;
                    trans : event, state → state;

        predicates  exp : event, state;
                    safe : state;

        axioms
             (* conversion of events *)
             tae1 = tae2 ↔ cta(tae1) = cta(tae2)),
             fbe ≠ fb_blankfalls → cfb(tae1) ≠ cta(tae2)),
             cfb(fb_blankfalls) = cta(ta_load),
             ....
             (* the transition function *)
                 fbe ≠ fb_blankfalls
             →     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)) ↔ fb_exp(fbe,fbs),

                 tae ≠ ta_load
             → exp(cta(tae),mkstate(fbs,tas,ros,prs,dbs,crs)) ↔ 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 impressing 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 = $(e_1, \ldots, e_n)$ 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

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

where control is the control procedure, trans is the transition function of the system, and 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 = $(e_1, \ldots, e_n)$, trans*(el,s) is inductively defined to be trans(e_n,trans(...trans(e_1,s) ...))). 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_a1loaded(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_a1loaded(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 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 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 safe(s) $\wedge$ safe-inv(s), our program will work in the sense that it will never reach an unsafe state.

To guarantee lifeness of the system, we must assure that in any state reached by the system we still have an expected event e, for which exp(e,s) holds, which means that not all devices are stopped. This is not the case for all states satisfying safe(s) and 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 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 lifeness, 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 lifeness 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 lifeness 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 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 ≠ ta_ro). A typical lemma used for simplification is:

```
   ta_loaded(ta_tr(ta_vto(ta_up),tas))
→         ta_tr(ta_vto(ta_up),tas) = taerr
   ∨ (    ta_tr(ta_vto(ta_up),tas)) ≠ taerr
        ∧ tas ≠ taerr ∧  ta_vpos(tas) = ta_down
        ∧ ta_hpos(tas) = ta_fb ∧  ta_loaded(tas))
```

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 lifeness 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 lifeness 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 lifeness 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 lifeness 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 lifeness 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 lifeness 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 lifeness assertion after about two weeks of work. The final version of the invariant

$$\text{safe}(s) \wedge \text{safe-inv}(s) \wedge \text{life-inv}(s)$$

can be found in appendix C. The statistic for the lifeness 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 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 safe-inv 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 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_tob1, 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 fbe_1: 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_tmvto (ta_tmvtos : ta_tposition)
      | ta_load | ta_unload
      | ta_at (ta_ats : ta_position)
      | ta_tat (ta_tats : ta_tposition)
      ;
  variables taev_1: ta_event; tapo_1: ta_position; tatpo_1: 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_aposition)
              | ro_a1load | ro_a1unload
              | ro_a2mv (ro_a2mvs : ro_aposition)
              | ro_a2load | ro_a2unload
              | ro_mvto (ro_mvtos : ro_position)
              | ro_a1at (ro_a1ats : ro_aposition)
              | ro_a2at (ro_a2ats : ro_aposition)
              | ro_at (ro_ats : ro_position)
              ;
    variables roe1: ro_event; ropo1: ro_position; roap1: ro_aposition;
end data specification

 PRESS_EVENTS =
data specification
    pr_position =  pr_up | pr_toup
                 | pr_down | pr_todown
                 | pr_mid | pr_tomid
                 ;
    pr_event =  pr_mvto (pr_mvtos : pr_position)
              | pr_load | pr_unload
              | pr_at (pr_ats : pr_position)
              ;
    variables pre1: pr_event; prpo1: pr_position;
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_aposition =  cr_aup | cr_atoup
                  | cr_adown | cr_atodown
                  | cr_amid | cr_atomid
                  ;
    cr_position =  cr_fb | cr_tofb
                 | cr_db | cr_todb
                 ;
    cr_event =  cr_amvto (cr_amvs : cr_aposition)
              | cr_mvto (cr_mvtos : cr_position)
              | cr_load | cr_unload
              | cr_aat (cr_aats : cr_aposition)
              | cr_at (cr_ats : cr_position)
              ;
    variables hge1: cr_event; hgpo1: cr_position; hgap1: cr_aposition;
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 $\to$ 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_2, fb_1: fb_state;
axioms
fb_state **generated by** fb_trans, fbinit;

$\mathrm{fb_errp}(\mathrm{fb}_1) \leftrightarrow \mathrm{fb}_1 = \mathrm{fberr}$,
$\mathrm{fbinit} \neq \mathrm{fberr}$,
$\mathrm{fb_pbinterrupted}(\mathrm{fb}_1) \wedge \neg\ \mathrm{fb_loaded}(\mathrm{fb}_1) \to \mathrm{fb}_1 = \mathrm{fberr}$,

$\mathrm{fb}_1 \neq \mathrm{fberr}$
$\to\ (\mathrm{fb}_1 = \mathrm{fb}_2$
$\leftrightarrow\ \mathrm{fb}_2 \neq \mathrm{fberr}$
$\wedge\ (\mathrm{fb_running}(\mathrm{fb}_1) \leftrightarrow \mathrm{fb_running}(\mathrm{fb}_2))$
$\wedge\ (\mathrm{fb_loaded}(\mathrm{fb}_1) \leftrightarrow \mathrm{fb_loaded}(\mathrm{fb}_2))$
$\wedge\ (\mathrm{fb_pbinterrupted}(\mathrm{fb}_1) \leftrightarrow \mathrm{fb_pbinterrupted}(\mathrm{fb}_2))$),

$\mathrm{fb_exp}(\mathrm{fb_sheetfalls}, \mathrm{fb}_1)$
$\leftrightarrow \mathrm{fb}_1 \neq \mathrm{fberr} \wedge \mathrm{fb_running}(\mathrm{fb}_1) \wedge \mathrm{fb_loaded}(\mathrm{fb}_1) \wedge \mathrm{fb_pbinterrupted}(\mathrm{fb}_1)$,
$\mathrm{fb_exp}(\mathrm{fb_pbinterruption}, \mathrm{fb}_1)$
$\leftrightarrow \mathrm{fb}_1 \neq \mathrm{fberr} \wedge \mathrm{fb_running}(\mathrm{fb}_1) \wedge \mathrm{fb_loaded}(\mathrm{fb}_1) \wedge \neg\ \mathrm{fb_pbinterrupted}(\mathrm{fb}_1)$,

$\neg\ \mathrm{fb_exp}(\mathrm{fb_on}, \mathrm{fb}_1)$,
$\neg\ \mathrm{fb_exp}(\mathrm{fb_off}, \mathrm{fb}_1)$,
$\neg\ \mathrm{fb_exp}(\mathrm{fb_load}, \mathrm{fb}_1)$,
$\neg\ \mathrm{fb_exp}(\mathrm{fb_unload}, \mathrm{fb}_1)$,

$\mathrm{fb}_1 \neq \mathrm{fberr} \wedge \neg\ \mathrm{fb_running}(\mathrm{fb}_1) \wedge \mathrm{fb}_2 = \mathrm{fb_trans}(\mathrm{fb_on}, \mathrm{fb}_1)$
$\to \mathrm{fb}_2 \neq \mathrm{fberr} \wedge \mathrm{fb_running}(\mathrm{fb}_2) \wedge$
$(\mathrm{fb_loaded}(\mathrm{fb}_2) \leftrightarrow \mathrm{fb_loaded}(\mathrm{fb}_1)) \wedge$
$(\mathrm{fb_pbinterrupted}(\mathrm{fb}_2) \leftrightarrow \mathrm{fb_pbinterrupted}(\mathrm{fb}_1))$,

$\mathrm{fb}_1 = \mathrm{fberr} \vee \mathrm{fb_running}(\mathrm{fb}_1)$
$\to \mathrm{fb_trans}(\mathrm{fb_on}, \mathrm{fb}_1) = \mathrm{fberr}$,

$\mathrm{fb}_1 \neq \mathrm{fberr} \wedge \mathrm{fb_running}(\mathrm{fb}_1) \wedge \mathrm{fb}_2 = \mathrm{fb_trans}(\mathrm{fb_off}, \mathrm{fb}_1)$
$\to \mathrm{fb}_2 \neq \mathrm{fberr} \wedge \neg\ \mathrm{fb_running}(\mathrm{fb}_2) \wedge$
$(\mathrm{fb_loaded}(\mathrm{fb}_2) \leftrightarrow \mathrm{fb_loaded}(\mathrm{fb}_1)) \wedge$
$(\mathrm{fb_pbinterrupted}(\mathrm{fb}_2) \leftrightarrow \mathrm{fb_pbinterrupted}(\mathrm{fb}_1))$,

$\mathrm{fb}_1 = \mathrm{fberr} \vee \neg\ \mathrm{fb_running}(\mathrm{fb}_1)$
$\to \mathrm{fb_trans}(\mathrm{fb_off}, \mathrm{fb}_1) = \mathrm{fberr}$,

$fb_1 \neq fberr \wedge fb_running(fb_1) \wedge$
$fb_loaded(fb_1) \wedge \neg\ fb_pbinterrupted(fb_1) \wedge fb_2 = fb_trans(fb_pbinterruption, fb_1)$
$\rightarrow fb_2 \neq fberr \wedge fb_running(fb_2) \wedge fb_loaded(fb_2) \wedge fb_pbinterrupted(fb_2),$

$fb_1 = fberr \vee \neg\ fb_running(fb_1) \vee \neg\ fb_loaded(fb_1) \vee fb_pbinterrupted(fb_1)$
$\rightarrow fb_trans(fb_pbinterruption, fb_1) = fberr,$

$fb_1 \neq fberr \wedge fb_running(fb_1) \wedge fb_loaded(fb_1) \wedge$
$fb_pbinterrupted(fb_1) \wedge fb_2 = fb_trans(fb_sheetfalls, fb_1)$
$\rightarrow fb_2 \neq fberr \wedge fb_running(fb_2) \wedge \neg\ fb_loaded(fb_2) \wedge \neg\ fb_pbinterrupted(fb_2),$

$fb_1 = fberr \vee \neg\ fb_running(fb_1) \vee \neg\ fb_loaded(fb_1) \vee \neg\ fb_pbinterrupted(fb_1)$
$\rightarrow fb_trans(fb_sheetfalls, fb_1) = fberr,$

$fb_1 \neq fberr \wedge \neg\ fb_loaded(fb_1) \wedge fb_2 = fb_trans(fb_load, fb_1)$
$\rightarrow fb_2 \neq fberr \wedge (fb_running(fb_2) \leftrightarrow fb_running(fb_1)) \wedge$
$fb_loaded(fb_2) \wedge \neg\ fb_pbinterrupted(fb_2),$

$fb_1 = fberr \vee fb_loaded(fb_1)$
$\rightarrow fb_trans(fb_load, fb_1) = fberr,$

$fb_1 \neq fberr \wedge fb_loaded(fb_1) \wedge$
$\neg\ fb_running(fb_1) \wedge fb_pbinterrupted(fb_1) \wedge fb_2 = fb_trans(fb_unload, fb_1)$
$\rightarrow fb_2 \neq fberr \wedge \neg\ fb_running(fb_2) \wedge \neg\ fb_loaded(fb_2) \wedge \neg\ fb_pbinterrupted(fb_2),$

$fb_1 = fberr \vee fb_running(fb_1) \vee \neg\ fb_loaded(fb_1) \vee \neg\ fb_pbinterrupted(fb_1)$
$\rightarrow fb_trans(fb_unload, fb_1) = fberr$

end 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_2, ta_1: ta_state; tae_1: ta_event;
axioms
ta_state **generated by** ta_trans, tainit;

$ta_errp(ta_1) \leftrightarrow ta_1 = taerr,$
$tainit \neq taerr,$
$ta_pos(ta_1) \neq ta_up \wedge ta_tpos(ta_1) \neq ta_fb \rightarrow ta_1 = taerr,$

$$
\begin{aligned}
& \mathrm{ta}_1 \neq \mathrm{taerr} \\
\rightarrow \quad & (\mathrm{ta}_1 = \mathrm{ta}_2 \\
& \leftrightarrow \quad \mathrm{ta}_2 \neq \mathrm{taerr} \\
& \quad \wedge \mathrm{ta_pos}(\mathrm{ta}_1) = \mathrm{ta_pos}(\mathrm{ta}_2) \\
& \quad \wedge \mathrm{ta_tpos}(\mathrm{ta}_1) = \mathrm{ta_tpos}(\mathrm{ta}_2) \\
& \quad \wedge (\mathrm{ta_loaded}(\mathrm{ta}_1) \leftrightarrow \mathrm{ta_loaded}(\mathrm{ta}_2))\ ),
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{ta_exp}(\mathrm{ta_at}(\mathrm{ta_up}), \mathrm{ta}_1) \\
\leftrightarrow\ & \mathrm{ta}_1 \neq \mathrm{taerr} \wedge \mathrm{ta_pos}(\mathrm{ta}_1) = \mathrm{ta_toup}, \\
& \mathrm{ta_exp}(\mathrm{ta_at}(\mathrm{ta_down}), \mathrm{ta}_1) \\
\leftrightarrow\ & \mathrm{ta}_1 \neq \mathrm{taerr} \wedge \mathrm{ta_pos}(\mathrm{ta}_1) = \mathrm{ta_todown}, \\
& \mathrm{ta_exp}(\mathrm{ta_tat}(\mathrm{ta_fb}), \mathrm{ta}_1) \\
\leftrightarrow\ & \mathrm{ta}_1 \neq \mathrm{taerr} \wedge \mathrm{ta_tpos}(\mathrm{ta}_1) = \mathrm{ta_tofb}, \\
& \mathrm{ta_exp}(\mathrm{ta_tat}(\mathrm{ta_ro}), \mathrm{ta}_1) \\
\leftrightarrow\ & \mathrm{ta}_1 \neq \mathrm{taerr} \wedge \mathrm{ta_tpos}(\mathrm{ta}_1) = \mathrm{ta_toro},
\end{aligned}
$$

$$
\begin{aligned}
& \neg\ \mathrm{ta_exp}(\mathrm{ta_mvto}(\mathrm{tapo}_1), \mathrm{ta}_1), \\
& \neg\ \mathrm{ta_exp}(\mathrm{ta_tmvto}(\mathrm{tatpo}_1), \mathrm{ta}_1), \\
& \neg\ \mathrm{ta_exp}(\mathrm{ta_load}, \mathrm{ta}_1), \\
& \neg\ \mathrm{ta_exp}(\mathrm{ta_unload}, \mathrm{ta}_1), \\
& \neg\ \mathrm{ta_exp}(\mathrm{ta_at}(\mathrm{ta_toup}), \mathrm{ta}_1), \\
& \neg\ \mathrm{ta_exp}(\mathrm{ta_at}(\mathrm{ta_todown}), \mathrm{ta}_1), \\
& \neg\ \mathrm{ta_exp}(\mathrm{ta_tat}(\mathrm{ta_tofb}), \mathrm{ta}_1), \\
& \neg\ \mathrm{ta_exp}(\mathrm{ta_tat}(\mathrm{ta_toro}), \mathrm{ta}_1),
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{ta}_1 \neq \mathrm{taerr} \wedge \mathrm{ta_pos}(\mathrm{ta}_1) = \mathrm{ta_down} \wedge \\
& \mathrm{ta_tpos}(\mathrm{ta}_1) = \mathrm{ta_fb} \wedge \mathrm{ta}_2 = \mathrm{ta_trans}(\mathrm{ta_mvto}(\mathrm{ta_up}), \mathrm{ta}_1) \\
\rightarrow\ & \mathrm{ta}_2 \neq \mathrm{taerr} \wedge \mathrm{ta_pos}(\mathrm{ta}_2) = \mathrm{ta_toup} \wedge \\
& \mathrm{ta_tpos}(\mathrm{ta}_2) = \mathrm{ta_fb} \wedge (\mathrm{ta_loaded}(\mathrm{ta}_2) \leftrightarrow \mathrm{ta_loaded}(\mathrm{ta}_1)),
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{ta}_1 = \mathrm{taerr} \vee \mathrm{ta_pos}(\mathrm{ta}_1) \neq \mathrm{ta_down} \vee \mathrm{ta_tpos}(\mathrm{ta}_1) \neq \mathrm{ta_fb} \\
\rightarrow\ & \mathrm{ta_trans}(\mathrm{ta_mvto}(\mathrm{ta_up}), \mathrm{ta}_1) = \mathrm{taerr},
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{ta}_1 \neq \mathrm{taerr} \wedge \mathrm{ta_pos}(\mathrm{ta}_1) = \mathrm{ta_up} \wedge \\
& \mathrm{ta_tpos}(\mathrm{ta}_1) = \mathrm{ta_fb} \wedge \mathrm{ta}_2 = \mathrm{ta_trans}(\mathrm{ta_mvto}(\mathrm{ta_down}), \mathrm{ta}_1) \\
\rightarrow\ & \mathrm{ta}_2 \neq \mathrm{taerr} \wedge \mathrm{ta_pos}(\mathrm{ta}_2) = \mathrm{ta_todown} \wedge \\
& \mathrm{ta_tpos}(\mathrm{ta}_2) = \mathrm{ta_fb} \wedge (\mathrm{ta_loaded}(\mathrm{ta}_2) \leftrightarrow \mathrm{ta_loaded}(\mathrm{ta}_1)),
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{ta}_1 = \mathrm{taerr} \vee \mathrm{ta_pos}(\mathrm{ta}_1) \neq \mathrm{ta_up} \vee \mathrm{ta_tpos}(\mathrm{ta}_1) \neq \mathrm{ta_fb} \\
\rightarrow\ & \mathrm{ta_trans}(\mathrm{ta_mvto}(\mathrm{ta_down}), \mathrm{ta}_1) = \mathrm{taerr},
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{ta}_1 \neq \mathrm{taerr} \wedge \mathrm{ta_pos}(\mathrm{ta}_1) = \mathrm{ta_up} \wedge \\
& \mathrm{ta_tpos}(\mathrm{ta}_1) = \mathrm{ta_ro} \wedge \mathrm{ta}_2 = \mathrm{ta_trans}(\mathrm{ta_tmvto}(\mathrm{ta_fb}), \mathrm{ta}_1) \\
\rightarrow\ & \mathrm{ta}_2 \neq \mathrm{taerr} \wedge \mathrm{ta_pos}(\mathrm{ta}_2) = \mathrm{ta_up} \wedge \\
& \mathrm{ta_tpos}(\mathrm{ta}_2) = \mathrm{ta_tofb} \wedge (\mathrm{ta_loaded}(\mathrm{ta}_2) \leftrightarrow \mathrm{ta_loaded}(\mathrm{ta}_1)),
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{ta}_1 = \mathrm{taerr} \vee \mathrm{ta_pos}(\mathrm{ta}_1) \neq \mathrm{ta_up} \vee \mathrm{ta_tpos}(\mathrm{ta}_1) \neq \mathrm{ta_ro} \\
\rightarrow\ & \mathrm{ta_trans}(\mathrm{ta_tmvto}(\mathrm{ta_fb}), \mathrm{ta}_1) = \mathrm{taerr},
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{ta}_1 \neq \mathrm{taerr} \wedge \mathrm{ta_pos}(\mathrm{ta}_1) = \mathrm{ta_up} \wedge \\
& \mathrm{ta_tpos}(\mathrm{ta}_1) = \mathrm{ta_fb} \wedge \mathrm{ta}_2 = \mathrm{ta_trans}(\mathrm{ta_tmvto}(\mathrm{ta_ro}), \mathrm{ta}_1) \\
\rightarrow\ & \mathrm{ta}_2 \neq \mathrm{taerr} \wedge \mathrm{ta_pos}(\mathrm{ta}_2) = \mathrm{ta_up} \wedge \\
& \mathrm{ta_tpos}(\mathrm{ta}_2) = \mathrm{ta_toro} \wedge (\mathrm{ta_loaded}(\mathrm{ta}_2) \leftrightarrow \mathrm{ta_loaded}(\mathrm{ta}_1)),
\end{aligned}
$$

$$\begin{array}{l} \quad \mathrm{ta_1 = taerr \lor ta_pos(ta_1) \neq ta_up \lor ta_tpos(ta_1) \neq ta_fb} \\ \to \mathrm{ta_trans(ta_tmvto(ta_ro), ta_1) = taerr}, \end{array}$$

$$\begin{array}{l} \quad \mathrm{ta_1 \neq taerr \land ta_pos(ta_1) = ta_toup} \land \\ \quad \mathrm{ta_tpos(ta_1) = ta_fb \land ta_2 = ta_trans(ta_at(ta_up), ta_1)} \\ \to \mathrm{ta_2 \neq taerr \land ta_pos(ta_2) = ta_up} \land \\ \quad \mathrm{ta_tpos(ta_2) = ta_fb \land (ta_loaded(ta_2) \leftrightarrow ta_loaded(ta_1))}, \end{array}$$

$$\begin{array}{l} \quad \mathrm{ta_1 = taerr \lor ta_pos(ta_1) \neq ta_toup \lor ta_tpos(ta_1) \neq ta_fb} \\ \to \mathrm{ta_trans(ta_at(ta_up), ta_1) = taerr}, \end{array}$$

$$\begin{array}{l} \quad \mathrm{ta_1 \neq taerr \land ta_pos(ta_1) = ta_todown} \land \\ \quad \mathrm{ta_tpos(ta_1) = ta_fb \land ta_2 = ta_trans(ta_at(ta_down), ta_1)} \\ \to \mathrm{ta_2 \neq taerr \land ta_pos(ta_2) = ta_down} \land \\ \quad \mathrm{ta_tpos(ta_2) = ta_fb \land (ta_loaded(ta_2) \leftrightarrow ta_loaded(ta_1))}, \end{array}$$

$$\begin{array}{l} \quad \mathrm{ta_1 = taerr \lor ta_pos(ta_1) \neq ta_todown \lor ta_tpos(ta_1) \neq ta_fb} \\ \to \mathrm{ta_trans(ta_at(ta_down), ta_1) = taerr}, \end{array}$$

$$\begin{array}{l} \quad \mathrm{ta_1 \neq taerr \land ta_pos(ta_1) = ta_up} \land \\ \quad \mathrm{ta_tpos(ta_1) = ta_tofb \land ta_2 = ta_trans(ta_tat(ta_fb), ta_1)} \\ \to \mathrm{ta_2 \neq taerr \land ta_pos(ta_2) = ta_up} \land \\ \quad \mathrm{ta_tpos(ta_2) = ta_fb \land (ta_loaded(ta_2) \leftrightarrow ta_loaded(ta_1))}, \end{array}$$

$$\begin{array}{l} \quad \mathrm{ta_1 = taerr \lor ta_pos(ta_1) \neq ta_up \lor ta_tpos(ta_1) \neq ta_tofb} \\ \to \mathrm{ta_trans(ta_tat(ta_fb), ta_1) = taerr}, \end{array}$$

$$\begin{array}{l} \quad \mathrm{ta_1 \neq taerr \land ta_pos(ta_1) = ta_up} \land \\ \quad \mathrm{ta_tpos(ta_1) = ta_toro \land ta_2 = ta_trans(ta_tat(ta_ro), ta_1)} \\ \to \mathrm{ta_2 \neq taerr \land ta_pos(ta_2) = ta_up} \land \\ \quad \mathrm{ta_tpos(ta_2) = ta_ro \land (ta_loaded(ta_2) \leftrightarrow ta_loaded(ta_1))}, \end{array}$$

$$\begin{array}{l} \quad \mathrm{ta_1 = taerr \lor ta_pos(ta_1) \neq ta_up \lor ta_tpos(ta_1) \neq ta_toro} \\ \to \mathrm{ta_trans(ta_tat(ta_ro), ta_1) = taerr}, \end{array}$$

$$\begin{array}{l} \quad \mathrm{ta_1 \neq taerr \land (ta_pos(ta_1) = ta_up \lor ta_pos(ta_1) = ta_down)} \land \\ \quad \mathrm{(ta_tpos(ta_1) = ta_ro \lor ta_tpos(ta_1) = ta_fb)} \land \\ \quad \mathrm{\neg\, ta_loaded(ta_1) \land ta_2 = ta_trans(ta_load, ta_1)} \\ \to \mathrm{ta_2 \neq taerr \land ta_pos(ta_2) = ta_pos(ta_1) \land ta_tpos(ta_2) = ta_tpos(ta_1) \land ta_loaded(ta_2)}, \end{array}$$

$$\begin{array}{l} \quad \mathrm{ta_1 = taerr \lor ta_pos(ta_1) \neq ta_up \land ta_pos(ta_1) \neq ta_down} \lor \\ \quad \mathrm{ta_tpos(ta_1) \neq ta_ro \land ta_tpos(ta_1) \neq ta_fb \lor ta_loaded(ta_1)} \\ \to \mathrm{ta_trans(ta_load, ta_1) = taerr}, \end{array}$$

$$\begin{array}{l} \quad \mathrm{ta_1 \neq taerr \land (ta_pos(ta_1) = ta_up \lor ta_pos(ta_1) = ta_down)} \land \\ \quad \mathrm{(ta_tpos(ta_1) = ta_ro \lor ta_tpos(ta_1) = ta_fb)} \land \\ \quad \mathrm{ta_loaded(ta_1) \land ta_2 = ta_trans(ta_unload, ta_1)} \\ \to \mathrm{ta_2 \neq taerr \land ta_pos(ta_2) = ta_pos(ta_1) \land ta_tpos(ta_2) = ta_tpos(ta_1) \land \neg\, ta_loaded(ta_2)}, \end{array}$$

$$\begin{array}{l} \quad \mathrm{ta_1 = taerr \lor ta_pos(ta_1) \neq ta_up \land ta_pos(ta_1) \neq ta_down} \lor \\ \quad \mathrm{ta_tpos(ta_1) \neq ta_ro \land ta_tpos(ta_1) \neq ta_fb \lor \neg\, ta_loaded(ta_1)} \\ \to \mathrm{ta_trans(ta_unload, ta_1) = taerr} \end{array}$$

end enrich

GENERIC_ROBOT =
enrich ROBOT_EVENTS **with**
sorts ro_state;
constants roerr : ro_state; roinit : ro_state;
functions

ro_a1pos	:	ro_state	$\longrightarrow$	ro_aposition	;
ro_a2pos	:	ro_state	$\longrightarrow$	ro_aposition	;
ro_pos	:	ro_state	$\longrightarrow$	ro_position	;
ro_trans	:	ro_event $\times$ ro_state	$\longrightarrow$	ro_state	;

predicates

ro_errp	:	ro_state;
ro_exp	:	ro_event $\times$ ro_state;
ro_a1loaded	:	ro_state;
ro_a2loaded	:	ro_state;

variables roe_1: ro_event; ro_4, ro_3, ro_2, ro_1, ro_0: ro_state;
axioms

ro_state **generated by** ro_trans, roinit;

$\text{ro_errp}(\text{ro}_1) \leftrightarrow \text{ro}_1 = \text{roerr}$,
$\text{roinit} \neq \text{roerr}$,
$(\text{ro_pos}(\text{ro}_1) = \text{ro_a1upr} \vee \text{ro_pos}(\text{ro}_1) = \text{ro_a2upr} \vee$
$\text{ro_pos}(\text{ro}_1) = \text{ro_a1tota} \vee \text{ro_pos}(\text{ro}_1) = \text{ro_a2todb})$
$\wedge (\text{ro_a1pos}(\text{ro}_1) \neq \text{ro_ain} \vee \text{ro_a2pos}(\text{ro}_1) \neq \text{ro_ain})$
$\rightarrow \text{ro}_1 = \text{roerr}$,

$\text{ro_pos}(\text{ro}_1) \neq \text{ro_a1ta} \wedge \text{ro_pos}(\text{ro}_1) \neq \text{ro_a1pr} \wedge \text{ro_a1pos}(\text{ro}_1) \neq \text{ro_ain} \rightarrow \text{ro}_1 = \text{roerr}$,
$\text{ro_pos}(\text{ro}_1) \neq \text{ro_a2db} \wedge \text{ro_pos}(\text{ro}_1) \neq \text{ro_a2pr} \wedge \text{ro_a2pos}(\text{ro}_1) \neq \text{ro_ain} \rightarrow \text{ro}_1 = \text{roerr}$,

$\text{ro}_1 \neq \text{roerr}$
$\rightarrow \quad (\text{ro}_1 = \text{ro}_2$
$\leftrightarrow \quad \text{ro}_2 \neq \text{roerr}$
$\wedge (\text{ro_a1loaded}(\text{ro}_1) \leftrightarrow \text{ro_a1loaded}(\text{ro}_2))$
$\wedge (\text{ro_a2loaded}(\text{ro}_1) \leftrightarrow \text{ro_a2loaded}(\text{ro}_2))$
$\wedge\ \text{ro_pos}(\text{ro}_1) = \text{ro_pos}(\text{ro}_2)$
$\wedge\ \text{ro_a1pos}(\text{ro}_1) = \text{ro_a1pos}(\text{ro}_2)$
$\wedge\ \text{ro_a2pos}(\text{ro}_1) = \text{ro_a2pos}(\text{ro}_2)\ )$,

$\text{ro_exp}(\text{ro_a1at}(\text{ro_ain}), \text{ro}_1) \leftrightarrow \text{ro}_1 \neq \text{roerr} \wedge \text{ro_a1pos}(\text{ro}_1) = \text{ro_atoin}$,
$\text{ro_exp}(\text{ro_a2at}(\text{ro_ain}), \text{ro}_1) \leftrightarrow \text{ro}_1 \neq \text{roerr} \wedge \text{ro_a2pos}(\text{ro}_1) = \text{ro_atoin}$,
$\text{ro_exp}(\text{ro_a1at}(\text{ro_amid}), \text{ro}_1) \leftrightarrow \text{ro}_1 \neq \text{roerr} \wedge \text{ro_a1pos}(\text{ro}_1) = \text{ro_atomid}$,
$\text{ro_exp}(\text{ro_a2at}(\text{ro_amid}), \text{ro}_1) \leftrightarrow \text{ro}_1 \neq \text{roerr} \wedge \text{ro_a2pos}(\text{ro}_1) = \text{ro_atomid}$,
$\text{ro_exp}(\text{ro_a1at}(\text{ro_aout}), \text{ro}_1) \leftrightarrow \text{ro}_1 \neq \text{roerr} \wedge \text{ro_a1pos}(\text{ro}_1) = \text{ro_atoout}$,
$\text{ro_exp}(\text{ro_a2at}(\text{ro_aout}), \text{ro}_1) \leftrightarrow \text{ro}_1 \neq \text{roerr} \wedge \text{ro_a2pos}(\text{ro}_1) = \text{ro_atoout}$,
$\text{ro_exp}(\text{ro_at}(\text{ro_a1ta}), \text{ro}_1) \leftrightarrow \text{ro}_1 \neq \text{roerr} \wedge \text{ro_pos}(\text{ro}_1) = \text{ro_a1tota}$,
$\text{ro_exp}(\text{ro_at}(\text{ro_a1pr}), \text{ro}_1) \leftrightarrow \text{ro}_1 \neq \text{roerr} \wedge \text{ro_pos}(\text{ro}_1) = \text{ro_a1upr}$,
$\text{ro_exp}(\text{ro_at}(\text{ro_a2db}), \text{ro}_1) \leftrightarrow \text{ro}_1 \neq \text{roerr} \wedge \text{ro_pos}(\text{ro}_1) = \text{ro_a2todb}$,
$\text{ro_exp}(\text{ro_at}(\text{ro_a2pr}), \text{ro}_1) \leftrightarrow \text{ro}_1 \neq \text{roerr} \wedge \text{ro_pos}(\text{ro}_1) = \text{ro_a2upr}$,

$\neg\ \text{ro_exp}(\text{ro_mvto}(\text{ropo}_1), \text{ro}_1)$,
$\neg\ \text{ro_exp}(\text{ro_a1load}, \text{ro}_1)$,
$\neg\ \text{ro_exp}(\text{ro_a2load}, \text{ro}_1)$,
$\neg\ \text{ro_exp}(\text{ro_a1unload}, \text{ro}_1)$,
$\neg\ \text{ro_exp}(\text{ro_a2unload}, \text{ro}_1)$,
$\neg\ \text{ro_exp}(\text{ro_at}(\text{ro_a1tota}), \text{ro}_1)$,

$\neg$ ro_exp(ro_at(ro_a1upr), ro_1),
$\neg$ ro_exp(ro_at(ro_a2todb), ro_1),
$\neg$ ro_exp(ro_at(ro_a2upr), ro_1),
$\neg$ ro_exp(ro_a1at(ro_atoin), ro_1),
$\neg$ ro_exp(ro_a1at(ro_atomid), ro_1),
$\neg$ ro_exp(ro_a1at(ro_atoout), ro_1),
$\neg$ ro_exp(ro_a2at(ro_atoin), ro_1),
$\neg$ ro_exp(ro_a2at(ro_atomid), ro_1),
$\neg$ ro_exp(ro_a2at(ro_atoout), ro_1),

$ro_1 \neq$ roerr $\wedge$ ro_pos(ro_1) = ro_a1upr $\wedge$ ro_a1pos(ro_1) = ro_ain $\wedge$
ro_a2pos(ro_1) = ro_ain $\wedge$ ro_2 = ro_trans(ro_at(ro_a1pr), ro_1)
$\rightarrow$ $ro_2 \neq$ roerr $\wedge$ ro_pos(ro_2) = ro_a1pr $\wedge$ ro_a1pos(ro_2) = ro_ain $\wedge$
ro_a2pos(ro_2) = ro_ain $\wedge$ (ro_a1loaded(ro_2) $\leftrightarrow$ ro_a1loaded(ro_1)) $\wedge$
(ro_a2loaded(ro_2) $\leftrightarrow$ ro_a2loaded(ro_1)),

ro_1 = roerr $\vee$ ro_pos(ro_1) $\neq$ ro_a1upr $\vee$
ro_a1pos(ro_1) $\neq$ ro_ain $\vee$ ro_a2pos(ro_1) $\neq$ ro_ain
$\rightarrow$ ro_trans(ro_at(ro_a1pr), ro_1) = roerr,

$ro_1 \neq$ roerr $\wedge$ ro_pos(ro_1) = ro_a1tota $\wedge$ ro_a1pos(ro_1) = ro_ain $\wedge$
ro_a2pos(ro_1) = ro_ain $\wedge$ ro_2 = ro_trans(ro_at(ro_a1ta), ro_1)
$\rightarrow$ $ro_2 \neq$ roerr $\wedge$ ro_pos(ro_2) = ro_a1ta $\wedge$ ro_a1pos(ro_2) = ro_ain $\wedge$
ro_a2pos(ro_2) = ro_ain $\wedge$ (ro_a1loaded(ro_2) $\leftrightarrow$ ro_a1loaded(ro_1)) $\wedge$
(ro_a2loaded(ro_2) $\leftrightarrow$ ro_a2loaded(ro_1)),

ro_1 = roerr $\vee$ ro_pos(ro_1) $\neq$ ro_a1tota $\vee$
ro_a1pos(ro_1) $\neq$ ro_ain $\vee$ ro_a2pos(ro_1) $\neq$ ro_ain
$\rightarrow$ ro_trans(ro_at(ro_a1ta), ro_1) = roerr,

$ro_1 \neq$ roerr $\wedge$ ro_pos(ro_1) = ro_a2upr $\wedge$ ro_a1pos(ro_1) = ro_ain $\wedge$
ro_a2pos(ro_1) = ro_ain $\wedge$ ro_2 = ro_trans(ro_at(ro_a2pr), ro_1)
$\rightarrow$ $ro_2 \neq$ roerr $\wedge$ ro_pos(ro_2) = ro_a2pr $\wedge$
ro_a1pos(ro_2) = ro_ain $\wedge$ ro_a2pos(ro_2) = ro_ain $\wedge$
(ro_a1loaded(ro_2) $\leftrightarrow$ ro_a1loaded(ro_1)) $\wedge$
(ro_a2loaded(ro_2) $\leftrightarrow$ ro_a2loaded(ro_1)),

ro_1 = roerr $\vee$ ro_pos(ro_1) $\neq$ ro_a2upr $\vee$
ro_a1pos(ro_1) $\neq$ ro_ain $\vee$ ro_a2pos(ro_1) $\neq$ ro_ain
$\rightarrow$ ro_trans(ro_at(ro_a2pr), ro_1) = roerr,

$ro_1 \neq$ roerr $\wedge$ ro_pos(ro_1) = ro_a2todb $\wedge$ ro_a1pos(ro_1) = ro_ain $\wedge$
ro_a2pos(ro_1) = ro_ain $\wedge$ ro_2 = ro_trans(ro_at(ro_a2db), ro_1)
$\rightarrow$ $ro_2 \neq$ roerr $\wedge$ ro_pos(ro_2) = ro_a2db $\wedge$ ro_a1pos(ro_2) = ro_ain $\wedge$
ro_a2pos(ro_2) = ro_ain $\wedge$ (ro_a1loaded(ro_2) $\leftrightarrow$ ro_a1loaded(ro_1)) $\wedge$
(ro_a2loaded(ro_2) $\leftrightarrow$ ro_a2loaded(ro_1)),

ro_1 = roerr $\vee$ ro_pos(ro_1) $\neq$ ro_a2todb $\vee$
ro_a1pos(ro_1) $\neq$ ro_ain $\vee$ ro_a2pos(ro_1) $\neq$ ro_ain
$\rightarrow$ ro_trans(ro_at(ro_a2db), ro_1) = roerr,

$$\begin{aligned}
&\mathrm{ro_1 \neq roerr} \wedge \\
&(\mathrm{ro_pos(ro_1) = ro_a1ta} \vee \mathrm{ro_pos(ro_1) = ro_a2pr} \vee \mathrm{ro_pos(ro_1) = ro_a2db}) \wedge \\
&\mathrm{ro_a1pos(ro_1) = ro_ain} \wedge \mathrm{ro_a2pos(ro_1) = ro_ain} \wedge \\
&\mathrm{ro_2 = ro_trans(ro_mvto(ro_a1pr), ro_1)} \\
\rightarrow\ &\mathrm{ro_2 \neq roerr} \wedge \mathrm{ro_pos(ro_2) = ro_a1upr} \wedge \mathrm{ro_a1pos(ro_2) = ro_ain} \wedge \\
&\mathrm{ro_a2pos(ro_2) = ro_ain} \wedge (\mathrm{ro_a1loaded(ro_2)} \leftrightarrow \mathrm{ro_a1loaded(ro_1)}) \wedge \\
&(\mathrm{ro_a2loaded(ro_2)} \leftrightarrow \mathrm{ro_a2loaded(ro_1)}),
\end{aligned}$$

$$\begin{aligned}
&\mathrm{ro_1 = roerr} \vee \\
&\mathrm{ro_pos(ro_1) \neq ro_a1ta} \wedge \mathrm{ro_pos(ro_1) \neq ro_a2pr} \wedge \mathrm{ro_pos(ro_1) \neq ro_a2db} \vee \\
&\mathrm{ro_a1pos(ro_1) \neq ro_ain} \vee \mathrm{ro_a2pos(ro_1) \neq ro_ain} \\
\rightarrow\ &\mathrm{ro_trans(ro_mvto(ro_a1pr), ro_1) = roerr},
\end{aligned}$$

$$\begin{aligned}
&\mathrm{ro_1 \neq roerr} \wedge \\
&(\mathrm{ro_pos(ro_1) = ro_a1pr} \vee \mathrm{ro_pos(ro_1) = ro_a2pr} \vee \mathrm{ro_pos(ro_1) = ro_a2db}) \wedge \\
&\mathrm{ro_a1pos(ro_1) = ro_ain} \wedge \mathrm{ro_a2pos(ro_1) = ro_ain} \wedge \\
&\mathrm{ro_2 = ro_trans(ro_mvto(ro_a1ta), ro_1)} \\
\rightarrow\ &\mathrm{ro_2 \neq roerr} \wedge \mathrm{ro_pos(ro_2) = ro_a1tota} \wedge \\
&\mathrm{ro_a1pos(ro_2) = ro_ain} \wedge \mathrm{ro_a2pos(ro_2) = ro_ain} \wedge \\
&(\mathrm{ro_a1loaded(ro_2)} \leftrightarrow \mathrm{ro_a1loaded(ro_1)}) \wedge \\
&(\mathrm{ro_a2loaded(ro_2)} \leftrightarrow \mathrm{ro_a2loaded(ro_1)}),
\end{aligned}$$

$$\begin{aligned}
&\mathrm{ro_1 = roerr} \vee \\
&\mathrm{ro_pos(ro_1) \neq ro_a1pr} \wedge \mathrm{ro_pos(ro_1) \neq ro_a2pr} \wedge \mathrm{ro_pos(ro_1) \neq ro_a2db} \vee \\
&\mathrm{ro_a1pos(ro_1) \neq ro_ain} \vee \mathrm{ro_a2pos(ro_1) \neq ro_ain} \\
\rightarrow\ &\mathrm{ro_trans(ro_mvto(ro_a1ta), ro_1) = roerr},
\end{aligned}$$

$$\begin{aligned}
&\mathrm{ro_1 \neq roerr} \wedge \\
&(\mathrm{ro_pos(ro_1) = ro_a1ta} \vee \mathrm{ro_pos(ro_1) = ro_a1pr} \vee \mathrm{ro_pos(ro_1) = ro_a2db}) \wedge \\
&\mathrm{ro_a1pos(ro_1) = ro_ain} \wedge \mathrm{ro_a2pos(ro_1) = ro_ain} \wedge \\
&\mathrm{ro_2 = ro_trans(ro_mvto(ro_a2pr), ro_1)} \\
\rightarrow\ &\mathrm{ro_2 \neq roerr} \wedge \mathrm{ro_pos(ro_2) = ro_a2upr} \wedge \\
&\mathrm{ro_a1pos(ro_2) = ro_ain} \wedge \mathrm{ro_a2pos(ro_2) = ro_ain} \wedge \\
&(\mathrm{ro_a1loaded(ro_2)} \leftrightarrow \mathrm{ro_a1loaded(ro_1)}) \wedge \\
&(\mathrm{ro_a2loaded(ro_2)} \leftrightarrow \mathrm{ro_a2loaded(ro_1)}),
\end{aligned}$$

$$\begin{aligned}
&\mathrm{ro_1 = roerr} \vee \\
&\mathrm{ro_pos(ro_1) \neq ro_a1ta} \wedge \mathrm{ro_pos(ro_1) \neq ro_a1pr} \wedge \mathrm{ro_pos(ro_1) \neq ro_a2db} \vee \\
&\mathrm{ro_a1pos(ro_1) \neq ro_ain} \vee \mathrm{ro_a2pos(ro_1) \neq ro_ain} \\
\rightarrow\ &\mathrm{ro_trans(ro_mvto(ro_a2pr), ro_1) = roerr},
\end{aligned}$$

$$\begin{aligned}
&\mathrm{ro_1 \neq roerr} \wedge \\
&(\mathrm{ro_pos(ro_1) = ro_a1ta} \vee \mathrm{ro_pos(ro_1) = ro_a1pr} \vee \mathrm{ro_pos(ro_1) = ro_a2pr}) \wedge \\
&\mathrm{ro_pos(ro_1) \neq ro_a2db} \wedge \mathrm{ro_a1pos(ro_1) = ro_ain} \wedge \mathrm{ro_a2pos(ro_1) = ro_ain} \wedge \\
&\mathrm{ro_2 = ro_trans(ro_mvto(ro_a2db), ro_1)} \\
\rightarrow\ &\mathrm{ro_2 \neq roerr} \wedge \mathrm{ro_pos(ro_2) = ro_a2todb} \wedge \\
&\mathrm{ro_a1pos(ro_2) = ro_ain} \wedge \mathrm{ro_a2pos(ro_2) = ro_ain} \wedge \\
&(\mathrm{ro_a1loaded(ro_2)} \leftrightarrow \mathrm{ro_a1loaded(ro_1)}) \wedge \\
&(\mathrm{ro_a2loaded(ro_2)} \leftrightarrow \mathrm{ro_a2loaded(ro_1)}),
\end{aligned}$$

$$\begin{aligned}
&\mathrm{ro_1 = roerr} \vee \\
&\mathrm{ro_pos(ro_1) \neq ro_a1ta} \wedge \mathrm{ro_pos(ro_1) \neq ro_a1pr} \wedge \mathrm{ro_pos(ro_1) \neq ro_a2pr} \vee \\
&\mathrm{ro_a1pos(ro_1) \neq ro_ain} \vee \mathrm{ro_a2pos(ro_1) \neq ro_ain} \\
\rightarrow\ &\mathrm{ro_trans(ro_mvto(ro_a2db), ro_1) = roerr},
\end{aligned}$$

$$
\begin{aligned}
& \mathrm{ro}_1 \neq \mathrm{roerr} \wedge (\mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a1ta} \vee \mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a1pr}) \wedge \\
& \mathrm{ro_a1pos}(\mathrm{ro}_1) = \mathrm{ro_atoin} \wedge \mathrm{ro}_2 = \mathrm{ro_trans}(\mathrm{ro_a1at}(\mathrm{ro_ain}), \mathrm{ro}_1) \\
\rightarrow\ & \mathrm{ro}_2 \neq \mathrm{roerr} \wedge \mathrm{ro_pos}(\mathrm{ro}_2) = \mathrm{ro_pos}(\mathrm{ro}_1) \wedge \\
& \mathrm{ro_a2pos}(\mathrm{ro}_2) = \mathrm{ro_a2pos}(\mathrm{ro}_1) \wedge \mathrm{ro_a1pos}(\mathrm{ro}_2) = \mathrm{ro_ain} \wedge \\
& (\mathrm{ro_a1loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a1loaded}(\mathrm{ro}_1)) \wedge \\
& (\mathrm{ro_a2loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a2loaded}(\mathrm{ro}_1)),
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{ro}_1 = \mathrm{roerr} \vee \mathrm{ro_a1pos}(\mathrm{ro}_1) \neq \mathrm{ro_atoin} \vee \\
& \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a1ta} \wedge \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a1pr} \\
\rightarrow\ & \mathrm{ro_trans}(\mathrm{ro_a1at}(\mathrm{ro_ain}), \mathrm{ro}_1) = \mathrm{roerr},
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{ro}_1 \neq \mathrm{roerr} \wedge (\mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a1ta} \vee \mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a1pr}) \wedge \\
& \mathrm{ro_a1pos}(\mathrm{ro}_1) = \mathrm{ro_atomid} \wedge \mathrm{ro}_2 = \mathrm{ro_trans}(\mathrm{ro_a1at}(\mathrm{ro_amid}), \mathrm{ro}_1) \\
\rightarrow\ & \mathrm{ro}_2 \neq \mathrm{roerr} \wedge \mathrm{ro_pos}(\mathrm{ro}_2) = \mathrm{ro_pos}(\mathrm{ro}_1) \wedge \\
& \mathrm{ro_a2pos}(\mathrm{ro}_2) = \mathrm{ro_a2pos}(\mathrm{ro}_1) \wedge \mathrm{ro_a1pos}(\mathrm{ro}_2) = \mathrm{ro_amid} \wedge \\
& (\mathrm{ro_a1loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a1loaded}(\mathrm{ro}_1)) \wedge \\
& (\mathrm{ro_a2loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a2loaded}(\mathrm{ro}_1)),
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{ro}_1 = \mathrm{roerr} \vee \mathrm{ro_a1pos}(\mathrm{ro}_1) \neq \mathrm{ro_atomid} \vee \\
& \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a1ta} \wedge \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a1pr} \\
\rightarrow\ & \mathrm{ro_trans}(\mathrm{ro_a1at}(\mathrm{ro_amid}), \mathrm{ro}_1) = \mathrm{roerr},
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{ro}_1 \neq \mathrm{roerr} \wedge (\mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a1ta} \vee \mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a1pr}) \wedge \\
& \mathrm{ro_a1pos}(\mathrm{ro}_1) = \mathrm{ro_atoout} \wedge \mathrm{ro}_2 = \mathrm{ro_trans}(\mathrm{ro_a1at}(\mathrm{ro_aout}), \mathrm{ro}_1) \\
\rightarrow\ & \mathrm{ro}_2 \neq \mathrm{roerr} \wedge \mathrm{ro_pos}(\mathrm{ro}_2) = \mathrm{ro_pos}(\mathrm{ro}_1) \wedge \\
& \mathrm{ro_a2pos}(\mathrm{ro}_2) = \mathrm{ro_a2pos}(\mathrm{ro}_1) \wedge \mathrm{ro_a1pos}(\mathrm{ro}_2) = \mathrm{ro_aout} \wedge \\
& (\mathrm{ro_a1loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a1loaded}(\mathrm{ro}_1)) \wedge \\
& (\mathrm{ro_a2loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a2loaded}(\mathrm{ro}_1)),
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{ro}_1 = \mathrm{roerr} \vee \mathrm{ro_a1pos}(\mathrm{ro}_1) \neq \mathrm{ro_atoout} \vee \\
& \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a1ta} \wedge \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a1pr} \\
\rightarrow\ & \mathrm{ro_trans}(\mathrm{ro_a1at}(\mathrm{ro_aout}), \mathrm{ro}_1) = \mathrm{roerr},
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{ro}_1 \neq \mathrm{roerr} \wedge (\mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a2pr} \vee \mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a2db}) \wedge \\
& \mathrm{ro_a2pos}(\mathrm{ro}_1) = \mathrm{ro_atoin} \wedge \mathrm{ro}_2 = \mathrm{ro_trans}(\mathrm{ro_a2at}(\mathrm{ro_ain}), \mathrm{ro}_1) \\
\rightarrow\ & \mathrm{ro}_2 \neq \mathrm{roerr} \wedge \mathrm{ro_pos}(\mathrm{ro}_2) = \mathrm{ro_pos}(\mathrm{ro}_1) \wedge \\
& \mathrm{ro_a1pos}(\mathrm{ro}_2) = \mathrm{ro_a1pos}(\mathrm{ro}_1) \wedge \mathrm{ro_a2pos}(\mathrm{ro}_2) = \mathrm{ro_ain} \wedge \\
& (\mathrm{ro_a1loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a1loaded}(\mathrm{ro}_1)) \wedge \\
& (\mathrm{ro_a2loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a2loaded}(\mathrm{ro}_1)),
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{ro}_1 = \mathrm{roerr} \vee \mathrm{ro_a2pos}(\mathrm{ro}_1) \neq \mathrm{ro_atoin} \vee \\
& \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a2pr} \wedge \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a2db} \\
\rightarrow\ & \mathrm{ro_trans}(\mathrm{ro_a2at}(\mathrm{ro_ain}), \mathrm{ro}_1) = \mathrm{roerr},
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{ro}_1 \neq \mathrm{roerr} \wedge (\mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a2pr} \vee \mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a2db}) \wedge \\
& \mathrm{ro_a2pos}(\mathrm{ro}_1) = \mathrm{ro_atomid} \wedge \\
& \mathrm{ro}_2 = \mathrm{ro_trans}(\mathrm{ro_a2at}(\mathrm{ro_amid}), \mathrm{ro}_1) \\
\rightarrow\ & \mathrm{ro}_2 \neq \mathrm{roerr} \wedge \mathrm{ro_pos}(\mathrm{ro}_2) = \mathrm{ro_pos}(\mathrm{ro}_1) \wedge \\
& \mathrm{ro_a1pos}(\mathrm{ro}_2) = \mathrm{ro_a1pos}(\mathrm{ro}_1) \wedge \mathrm{ro_a2pos}(\mathrm{ro}_2) = \mathrm{ro_amid} \wedge \\
& (\mathrm{ro_a1loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a1loaded}(\mathrm{ro}_1)) \wedge \\
& (\mathrm{ro_a2loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a2loaded}(\mathrm{ro}_1)),
\end{aligned}
$$

$$\begin{aligned}
&\mathrm{ro}_1 = \mathrm{roerr} \vee \mathrm{ro_a2pos}(\mathrm{ro}_1) \neq \mathrm{ro_atomid} \vee \\
&\mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a2pr} \wedge \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a2db} \\
\rightarrow\ &\mathrm{ro_trans}(\mathrm{ro_a2at}(\mathrm{ro_amid}), \mathrm{ro}_1) = \mathrm{roerr},
\end{aligned}$$

$$\begin{aligned}
&\mathrm{ro}_1 \neq \mathrm{roerr} \wedge (\mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a2pr} \vee \mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a2db}) \wedge \\
&\mathrm{ro_a2pos}(\mathrm{ro}_1) = \mathrm{ro_atoout} \wedge \mathrm{ro}_2 = \mathrm{ro_trans}(\mathrm{ro_a2at}(\mathrm{ro_aout}), \mathrm{ro}_1) \\
\rightarrow\ &\mathrm{ro}_2 \neq \mathrm{roerr} \wedge \mathrm{ro_pos}(\mathrm{ro}_2) = \mathrm{ro_pos}(\mathrm{ro}_1) \wedge \\
&\mathrm{ro_a1pos}(\mathrm{ro}_2) = \mathrm{ro_a1pos}(\mathrm{ro}_1) \wedge \mathrm{ro_a2pos}(\mathrm{ro}_2) = \mathrm{ro_aout} \wedge \\
&(\mathrm{ro_a1loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a1loaded}(\mathrm{ro}_1)) \wedge \\
&(\mathrm{ro_a2loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a2loaded}(\mathrm{ro}_1)),
\end{aligned}$$

$$\begin{aligned}
&\mathrm{ro}_1 = \mathrm{roerr} \vee \mathrm{ro_a2pos}(\mathrm{ro}_1) \neq \mathrm{ro_atoout} \vee \\
&\mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a2pr} \wedge \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a2db} \\
\rightarrow\ &\mathrm{ro_trans}(\mathrm{ro_a2at}(\mathrm{ro_aout}), \mathrm{ro}_1) = \mathrm{roerr},
\end{aligned}$$

$$\begin{aligned}
&\mathrm{ro}_1 \neq \mathrm{roerr} \wedge (\mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a1ta} \vee \mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a1pr}) \wedge \\
&(\mathrm{ro_a1pos}(\mathrm{ro}_1) = \mathrm{ro_amid} \vee \mathrm{ro_a1pos}(\mathrm{ro}_1) = \mathrm{ro_aout}) \wedge \\
&\mathrm{ro}_2 = \mathrm{ro_trans}(\mathrm{ro_a1mv}(\mathrm{ro_ain}), \mathrm{ro}_1) \\
\rightarrow\ &\mathrm{ro}_2 \neq \mathrm{roerr} \wedge \mathrm{ro_pos}(\mathrm{ro}_2) = \mathrm{ro_pos}(\mathrm{ro}_1) \wedge \\
&\mathrm{ro_a2pos}(\mathrm{ro}_2) = \mathrm{ro_a2pos}(\mathrm{ro}_1) \wedge \mathrm{ro_a1pos}(\mathrm{ro}_2) = \mathrm{ro_atoin} \wedge \\
&(\mathrm{ro_a1loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a1loaded}(\mathrm{ro}_1)) \wedge \\
&(\mathrm{ro_a2loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a2loaded}(\mathrm{ro}_1)),
\end{aligned}$$

$$\begin{aligned}
&\mathrm{ro}_1 = \mathrm{roerr} \vee \\
&\mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a1ta} \wedge \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a1pr} \vee \\
&\mathrm{ro_a1pos}(\mathrm{ro}_1) \neq \mathrm{ro_amid} \wedge \mathrm{ro_a1pos}(\mathrm{ro}_1) \neq \mathrm{ro_aout} \\
\rightarrow\ &\mathrm{ro_trans}(\mathrm{ro_a1mv}(\mathrm{ro_ain}), \mathrm{ro}_1) = \mathrm{roerr},
\end{aligned}$$

$$\begin{aligned}
&\mathrm{ro}_1 \neq \mathrm{roerr} \wedge (\mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a1ta} \vee \mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a1pr}) \wedge \\
&(\mathrm{ro_a1pos}(\mathrm{ro}_1) = \mathrm{ro_ain} \vee \mathrm{ro_a1pos}(\mathrm{ro}_1) = \mathrm{ro_aout}) \wedge \\
&\mathrm{ro}_2 = \mathrm{ro_trans}(\mathrm{ro_a1mv}(\mathrm{ro_amid}), \mathrm{ro}_1) \\
\rightarrow\ &\mathrm{ro}_2 \neq \mathrm{roerr} \wedge \mathrm{ro_pos}(\mathrm{ro}_2) = \mathrm{ro_pos}(\mathrm{ro}_1) \wedge \mathrm{ro_a2pos}(\mathrm{ro}_2) = \mathrm{ro_a2pos}(\mathrm{ro}_1) \wedge \\
&\mathrm{ro_a1pos}(\mathrm{ro}_2) = \mathrm{ro_atomid} \wedge (\mathrm{ro_a1loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a1loaded}(\mathrm{ro}_1)) \wedge \\
&(\mathrm{ro_a2loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a2loaded}(\mathrm{ro}_1)),
\end{aligned}$$

$$\begin{aligned}
&\mathrm{ro}_1 = \mathrm{roerr} \vee \\
&\mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a1ta} \wedge \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a1pr} \vee \\
&\mathrm{ro_a1pos}(\mathrm{ro}_1) \neq \mathrm{ro_ain} \wedge \mathrm{ro_a1pos}(\mathrm{ro}_1) \neq \mathrm{ro_aout} \\
\rightarrow\ &\mathrm{ro_trans}(\mathrm{ro_a1mv}(\mathrm{ro_amid}), \mathrm{ro}_1) = \mathrm{roerr},
\end{aligned}$$

$$\begin{aligned}
&\mathrm{ro}_1 \neq \mathrm{roerr} \wedge (\mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a1ta} \vee \mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a1pr}) \wedge \\
&(\mathrm{ro_a1pos}(\mathrm{ro}_1) = \mathrm{ro_ain} \vee \mathrm{ro_a1pos}(\mathrm{ro}_1) = \mathrm{ro_amid}) \wedge \\
&\mathrm{ro}_2 = \mathrm{ro_trans}(\mathrm{ro_a1mv}(\mathrm{ro_aout}), \mathrm{ro}_1) \\
\rightarrow\ &\mathrm{ro}_2 \neq \mathrm{roerr} \wedge \mathrm{ro_pos}(\mathrm{ro}_2) = \mathrm{ro_pos}(\mathrm{ro}_1) \wedge \mathrm{ro_a2pos}(\mathrm{ro}_2) = \mathrm{ro_a2pos}(\mathrm{ro}_1) \wedge \\
&\mathrm{ro_a1pos}(\mathrm{ro}_2) = \mathrm{ro_atoout} \wedge (\mathrm{ro_a1loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a1loaded}(\mathrm{ro}_1)) \wedge \\
&(\mathrm{ro_a2loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a2loaded}(\mathrm{ro}_1)),
\end{aligned}$$

$$\begin{aligned}
&\mathrm{ro}_1 = \mathrm{roerr} \vee \\
&\mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a1ta} \wedge \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a1pr} \vee \\
&\mathrm{ro_a1pos}(\mathrm{ro}_1) \neq \mathrm{ro_ain} \wedge \mathrm{ro_a1pos}(\mathrm{ro}_1) \neq \mathrm{ro_amid} \\
\rightarrow\ &\mathrm{ro_trans}(\mathrm{ro_a1mv}(\mathrm{ro_aout}), \mathrm{ro}_1) = \mathrm{roerr},
\end{aligned}$$

$$\begin{array}{ll}
 & \mathrm{ro}_1 \neq \mathrm{roerr} \wedge (\mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a2pr} \vee \mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a2db}) \wedge \\
 & (\mathrm{ro_a2pos}(\mathrm{ro}_1) = \mathrm{ro_amid} \vee \mathrm{ro_a2pos}(\mathrm{ro}_1) = \mathrm{ro_aout}) \wedge \\
 & \mathrm{ro}_2 = \mathrm{ro_trans}(\mathrm{ro_a2mv}(\mathrm{ro_ain}), \mathrm{ro}_1) \\
\rightarrow & \mathrm{ro}_2 \neq \mathrm{roerr} \wedge \mathrm{ro_pos}(\mathrm{ro}_2) = \mathrm{ro_pos}(\mathrm{ro}_1) \wedge \\
 & \mathrm{ro_a1pos}(\mathrm{ro}_2) = \mathrm{ro_a1pos}(\mathrm{ro}_1) \wedge \mathrm{ro_a2pos}(\mathrm{ro}_2) = \mathrm{ro_atoin} \wedge \\
 & (\mathrm{ro_a1loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a1loaded}(\mathrm{ro}_1)) \wedge \\
 & (\mathrm{ro_a2loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a2loaded}(\mathrm{ro}_1)),
\end{array}$$

$$\begin{array}{ll}
 & \mathrm{ro}_1 = \mathrm{roerr} \vee \\
 & \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a2pr} \wedge \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a2db} \vee \\
 & \mathrm{ro_a2pos}(\mathrm{ro}_1) \neq \mathrm{ro_amid} \wedge \mathrm{ro_a2pos}(\mathrm{ro}_1) \neq \mathrm{ro_aout} \\
\rightarrow & \mathrm{ro_trans}(\mathrm{ro_a2mv}(\mathrm{ro_ain}), \mathrm{ro}_1) = \mathrm{roerr},
\end{array}$$

$$\begin{array}{ll}
 & \mathrm{ro}_1 \neq \mathrm{roerr} \wedge (\mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a2pr} \vee \mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a2db}) \wedge \\
 & (\mathrm{ro_a2pos}(\mathrm{ro}_1) = \mathrm{ro_ain} \vee \mathrm{ro_a2pos}(\mathrm{ro}_1) = \mathrm{ro_aout}) \wedge \\
 & \mathrm{ro}_2 = \mathrm{ro_trans}(\mathrm{ro_a2mv}(\mathrm{ro_amid}), \mathrm{ro}_1) \\
\rightarrow & \mathrm{ro}_2 \neq \mathrm{roerr} \wedge \mathrm{ro_pos}(\mathrm{ro}_2) = \mathrm{ro_pos}(\mathrm{ro}_1) \wedge \\
 & \mathrm{ro_a1pos}(\mathrm{ro}_2) = \mathrm{ro_a1pos}(\mathrm{ro}_1) \wedge \mathrm{ro_a2pos}(\mathrm{ro}_2) = \mathrm{ro_atomid} \wedge \\
 & (\mathrm{ro_a1loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a1loaded}(\mathrm{ro}_1)) \wedge \\
 & (\mathrm{ro_a2loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a2loaded}(\mathrm{ro}_1)),
\end{array}$$

$$\begin{array}{ll}
 & \mathrm{ro}_1 = \mathrm{roerr} \vee \\
 & \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a2pr} \wedge \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a2db} \vee \\
 & \mathrm{ro_a2pos}(\mathrm{ro}_1) \neq \mathrm{ro_ain} \wedge \mathrm{ro_a2pos}(\mathrm{ro}_1) \neq \mathrm{ro_aout} \\
\rightarrow & \mathrm{ro_trans}(\mathrm{ro_a2mv}(\mathrm{ro_amid}), \mathrm{ro}_1) = \mathrm{roerr},
\end{array}$$

$$\begin{array}{ll}
 & \mathrm{ro}_1 \neq \mathrm{roerr} \wedge (\mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a2pr} \vee \mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a2db}) \wedge \\
 & (\mathrm{ro_a2pos}(\mathrm{ro}_1) = \mathrm{ro_ain} \vee \mathrm{ro_a2pos}(\mathrm{ro}_1) = \mathrm{ro_amid}) \wedge \\
 & \mathrm{ro}_2 = \mathrm{ro_trans}(\mathrm{ro_a2mv}(\mathrm{ro_aout}), \mathrm{ro}_1) \\
\rightarrow & \mathrm{ro}_2 \neq \mathrm{roerr} \wedge \mathrm{ro_pos}(\mathrm{ro}_2) = \mathrm{ro_pos}(\mathrm{ro}_1) \wedge \\
 & \mathrm{ro_a1pos}(\mathrm{ro}_2) = \mathrm{ro_a1pos}(\mathrm{ro}_1) \wedge \mathrm{ro_a2pos}(\mathrm{ro}_2) = \mathrm{ro_atoout} \wedge \\
 & (\mathrm{ro_a1loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a1loaded}(\mathrm{ro}_1)) \wedge \\
 & (\mathrm{ro_a2loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a2loaded}(\mathrm{ro}_1)),
\end{array}$$

$$\begin{array}{ll}
 & \mathrm{ro}_1 = \mathrm{roerr} \vee \\
 & \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a2pr} \wedge \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a2db} \vee \\
 & \mathrm{ro_a2pos}(\mathrm{ro}_1) \neq \mathrm{ro_ain} \wedge \mathrm{ro_a2pos}(\mathrm{ro}_1) \neq \mathrm{ro_amid} \\
\rightarrow & \mathrm{ro_trans}(\mathrm{ro_a2mv}(\mathrm{ro_aout}), \mathrm{ro}_1) = \mathrm{roerr},
\end{array}$$

$$\begin{array}{ll}
 & \mathrm{ro}_1 \neq \mathrm{roerr} \wedge (\mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a1ta} \vee \mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a1pr}) \wedge \\
 & (\mathrm{ro_a1pos}(\mathrm{ro}_1) = \mathrm{ro_amid} \vee \mathrm{ro_a1pos}(\mathrm{ro}_1) = \mathrm{ro_aout}) \wedge \\
 & \neg\, \mathrm{ro_a1loaded}(\mathrm{ro}_1) \wedge \mathrm{ro}_2 = \mathrm{ro_trans}(\mathrm{ro_a1load}, \mathrm{ro}_1) \\
\rightarrow & \mathrm{ro}_2 \neq \mathrm{roerr} \wedge \mathrm{ro_pos}(\mathrm{ro}_2) = \mathrm{ro_pos}(\mathrm{ro}_1) \wedge \\
 & \mathrm{ro_a1pos}(\mathrm{ro}_2) = \mathrm{ro_a1pos}(\mathrm{ro}_1) \wedge \mathrm{ro_a2pos}(\mathrm{ro}_2) = \mathrm{ro_a2pos}(\mathrm{ro}_1) \wedge \\
 & \mathrm{ro_a1loaded}(\mathrm{ro}_2) \wedge (\mathrm{ro_a2loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a2loaded}(\mathrm{ro}_1)),
\end{array}$$

$$\begin{array}{ll}
 & \mathrm{ro}_1 = \mathrm{roerr} \vee \\
 & \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a1ta} \wedge \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a1pr} \vee \\
 & \mathrm{ro_a1pos}(\mathrm{ro}_1) \neq \mathrm{ro_amid} \wedge \mathrm{ro_a1pos}(\mathrm{ro}_1) \neq \mathrm{ro_aout} \vee \mathrm{ro_a1loaded}(\mathrm{ro}_1) \\
\rightarrow & \mathrm{ro_trans}(\mathrm{ro_a1load}, \mathrm{ro}_1) = \mathrm{roerr},
\end{array}$$

$$
\begin{array}{l}
\quad \mathrm{ro}_1 \neq \mathrm{roerr} \wedge (\mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a1ta} \vee \mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a1pr}) \wedge \\
\quad (\mathrm{ro_a1pos}(\mathrm{ro}_1) = \mathrm{ro_amid} \vee \mathrm{ro_a1pos}(\mathrm{ro}_1) = \mathrm{ro_aout}) \wedge \\
\quad \mathrm{ro_a1loaded}(\mathrm{ro}_1) \wedge \mathrm{ro}_2 = \mathrm{ro_trans}(\mathrm{ro_a1unload}, \mathrm{ro}_1) \\
\rightarrow \mathrm{ro}_2 \neq \mathrm{roerr} \wedge \mathrm{ro_pos}(\mathrm{ro}_2) = \mathrm{ro_pos}(\mathrm{ro}_1) \wedge \\
\quad \mathrm{ro_a1pos}(\mathrm{ro}_2) = \mathrm{ro_a1pos}(\mathrm{ro}_1) \wedge \mathrm{ro_a2pos}(\mathrm{ro}_2) = \mathrm{ro_a2pos}(\mathrm{ro}_1) \wedge \\
\quad \neg\, \mathrm{ro_a1loaded}(\mathrm{ro}_2) \wedge (\mathrm{ro_a2loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a2loaded}(\mathrm{ro}_1)),
\end{array}
$$

$$
\begin{array}{l}
\quad \mathrm{ro}_1 = \mathrm{roerr} \vee \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a1ta} \wedge \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a1pr} \vee \\
\quad \mathrm{ro_a1pos}(\mathrm{ro}_1) \neq \mathrm{ro_amid} \wedge \mathrm{ro_a1pos}(\mathrm{ro}_1) \neq \mathrm{ro_aout} \vee \neg\, \mathrm{ro_a1loaded}(\mathrm{ro}_1) \\
\rightarrow \mathrm{ro_trans}(\mathrm{ro_a1unload}, \mathrm{ro}_1) = \mathrm{roerr},
\end{array}
$$

$$
\begin{array}{l}
\quad \mathrm{ro}_1 \neq \mathrm{roerr} \wedge (\mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a2db} \vee \mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a2pr}) \wedge \\
\quad (\mathrm{ro_a2pos}(\mathrm{ro}_1) = \mathrm{ro_amid} \vee \mathrm{ro_a2pos}(\mathrm{ro}_1) = \mathrm{ro_aout}) \wedge \\
\quad \neg\, \mathrm{ro_a2loaded}(\mathrm{ro}_1) \wedge \mathrm{ro}_2 = \mathrm{ro_trans}(\mathrm{ro_a2load}, \mathrm{ro}_1) \\
\rightarrow \mathrm{ro}_2 \neq \mathrm{roerr} \wedge \mathrm{ro_pos}(\mathrm{ro}_2) = \mathrm{ro_pos}(\mathrm{ro}_1) \wedge \\
\quad \mathrm{ro_a1pos}(\mathrm{ro}_2) = \mathrm{ro_a1pos}(\mathrm{ro}_1) \wedge \mathrm{ro_a2pos}(\mathrm{ro}_2) = \mathrm{ro_a2pos}(\mathrm{ro}_1) \wedge \\
\quad \mathrm{ro_a2loaded}(\mathrm{ro}_2) \wedge (\mathrm{ro_a1loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a1loaded}(\mathrm{ro}_1)),
\end{array}
$$

$$
\begin{array}{l}
\quad \mathrm{ro}_1 = \mathrm{roerr} \vee \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a2db} \wedge \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a2pr} \vee \\
\quad \mathrm{ro_a2pos}(\mathrm{ro}_1) \neq \mathrm{ro_amid} \wedge \mathrm{ro_a2pos}(\mathrm{ro}_1) \neq \mathrm{ro_aout} \vee \mathrm{ro_a2loaded}(\mathrm{ro}_1) \\
\rightarrow \mathrm{ro_trans}(\mathrm{ro_a2load}, \mathrm{ro}_1) = \mathrm{roerr},
\end{array}
$$

$$
\begin{array}{l}
\quad \mathrm{ro}_1 \neq \mathrm{roerr} \wedge (\mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a2db} \vee \mathrm{ro_pos}(\mathrm{ro}_1) = \mathrm{ro_a2pr}) \wedge \\
\quad (\mathrm{ro_a2pos}(\mathrm{ro}_1) = \mathrm{ro_amid} \vee \mathrm{ro_a2pos}(\mathrm{ro}_1) = \mathrm{ro_aout}) \wedge \\
\quad \mathrm{ro_a2loaded}(\mathrm{ro}_1) \wedge \mathrm{ro}_2 = \mathrm{ro_trans}(\mathrm{ro_a2unload}, \mathrm{ro}_1) \\
\rightarrow \mathrm{ro}_2 \neq \mathrm{roerr} \wedge \mathrm{ro_pos}(\mathrm{ro}_2) = \mathrm{ro_pos}(\mathrm{ro}_1) \wedge \\
\quad \mathrm{ro_a1pos}(\mathrm{ro}_2) = \mathrm{ro_a1pos}(\mathrm{ro}_1) \wedge \mathrm{ro_a2pos}(\mathrm{ro}_2) = \mathrm{ro_a2pos}(\mathrm{ro}_1) \wedge \\
\quad \neg\, \mathrm{ro_a2loaded}(\mathrm{ro}_2) \wedge (\mathrm{ro_a1loaded}(\mathrm{ro}_2) \leftrightarrow \mathrm{ro_a1loaded}(\mathrm{ro}_1)),
\end{array}
$$

$$
\begin{array}{l}
\quad \mathrm{ro}_1 = \mathrm{roerr} \vee \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a2db} \wedge \mathrm{ro_pos}(\mathrm{ro}_1) \neq \mathrm{ro_a2pr} \vee \\
\quad \mathrm{ro_a2pos}(\mathrm{ro}_1) \neq \mathrm{ro_amid} \wedge \mathrm{ro_a2pos}(\mathrm{ro}_1) \neq \mathrm{ro_aout} \vee \neg\, \mathrm{ro_a2loaded}(\mathrm{ro}_1) \\
\rightarrow \mathrm{ro_trans}(\mathrm{ro_a2unload}, \mathrm{ro}_1) = \mathrm{roerr},
\end{array}
$$

$$
\begin{array}{l}
\mathrm{ro_trans}(\mathrm{ro_at}(\mathrm{ro_a1tota}), \mathrm{ro}_1) = \mathrm{roerr}, \\
\mathrm{ro_trans}(\mathrm{ro_at}(\mathrm{ro_a1upr}), \mathrm{ro}_1) = \mathrm{roerr}, \\
\mathrm{ro_trans}(\mathrm{ro_at}(\mathrm{ro_a2upr}), \mathrm{ro}_1) = \mathrm{roerr}, \\
\mathrm{ro_trans}(\mathrm{ro_at}(\mathrm{ro_a2todb}), \mathrm{ro}_1) = \mathrm{roerr}, \\
\mathrm{ro_trans}(\mathrm{ro_mvto}(\mathrm{ro_a1tota}), \mathrm{ro}_1) = \mathrm{roerr}, \\
\mathrm{ro_trans}(\mathrm{ro_mvto}(\mathrm{ro_a1upr}), \mathrm{ro}_1) = \mathrm{roerr}, \\
\mathrm{ro_trans}(\mathrm{ro_mvto}(\mathrm{ro_a2upr}), \mathrm{ro}_1) = \mathrm{roerr}, \\
\mathrm{ro_trans}(\mathrm{ro_mvto}(\mathrm{ro_a2todb}), \mathrm{ro}_1) = \mathrm{roerr}, \\
\mathrm{ro_trans}(\mathrm{ro_a1mv}(\mathrm{ro_atoin}), \mathrm{ro}_1) = \mathrm{roerr}, \\
\mathrm{ro_trans}(\mathrm{ro_a1mv}(\mathrm{ro_atomid}), \mathrm{ro}_1) = \mathrm{roerr}, \\
\mathrm{ro_trans}(\mathrm{ro_a1mv}(\mathrm{ro_atoout}), \mathrm{ro}_1) = \mathrm{roerr}, \\
\mathrm{ro_trans}(\mathrm{ro_a2mv}(\mathrm{ro_atoin}), \mathrm{ro}_1) = \mathrm{roerr}, \\
\mathrm{ro_trans}(\mathrm{ro_a2mv}(\mathrm{ro_atomid}), \mathrm{ro}_1) = \mathrm{roerr}, \\
\mathrm{ro_trans}(\mathrm{ro_a2mv}(\mathrm{ro_atoout}), \mathrm{ro}_1) = \mathrm{roerr}, \\
\mathrm{ro_trans}(\mathrm{ro_a1at}(\mathrm{ro_atoin}), \mathrm{ro}_1) = \mathrm{roerr}, \\
\mathrm{ro_trans}(\mathrm{ro_a1at}(\mathrm{ro_atomid}), \mathrm{ro}_1) = \mathrm{roerr}, \\
\mathrm{ro_trans}(\mathrm{ro_a1at}(\mathrm{ro_atoout}), \mathrm{ro}_1) = \mathrm{roerr}, \\
\mathrm{ro_trans}(\mathrm{ro_a2at}(\mathrm{ro_atoin}), \mathrm{ro}_1) = \mathrm{roerr}, \\
\mathrm{ro_trans}(\mathrm{ro_a2at}(\mathrm{ro_atomid}), \mathrm{ro}_1) = \mathrm{roerr}, \\
\mathrm{ro_trans}(\mathrm{ro_a2at}(\mathrm{ro_atoout}), \mathrm{ro}_1) = \mathrm{roerr}
\end{array}
$$

end enrich

GENERIC_PRESS =
enrich PRESS_EVENTS **with**
sorts pr_state;
constants prerr : pr_state; prinit : pr_state;
functions

pr_pos	:	pr_state	$\rightarrow$	pr_position	;
pr_trans	:	pr_event $\times$ pr_state	$\rightarrow$	pr_state	;

predicates

pr_errp	:	pr_state;
pr_exp	:	pr_event $\times$ pr_state;
pr_pressed	:	pr_state;
pr_loaded	:	pr_state;

variables pre$_2$: pr_event; pr$_3$, pr$_2$, pr$_1$: pr_state;
axioms
pr_state **generated by** pr_trans, prinit;

$\text{pr_errp}(\text{pr}_1) \leftrightarrow \text{pr}_1 = \text{prerr},$
$\text{prinit} \neq \text{prerr},$

$\text{pr}_1 \neq \text{prerr}$
$\rightarrow \quad (\text{pr}_1 = \text{pr}_2$
$\leftrightarrow \quad \text{pr}_2 \neq \text{prerr}$
$\wedge\ (\text{pr_pressed}(\text{pr}_1) \leftrightarrow \text{pr_pressed}(\text{pr}_2))$
$\wedge\ (\text{pr_loaded}(\text{pr}_1) \leftrightarrow \text{pr_loaded}(\text{pr}_2))$
$\wedge\ \text{pr_pos}(\text{pr}_1) = \text{pr_pos}(\text{pr}_2)\ ),$

$\text{pr_exp}(\text{pr_at}(\text{pr_up}), \text{pr}_1) \leftrightarrow \text{pr}_1 \neq \text{prerr} \wedge \text{pr_pos}(\text{pr}_1) = \text{pr_toup},$
$\text{pr_exp}(\text{pr_at}(\text{pr_mid}), \text{pr}_1) \leftrightarrow \text{pr}_1 \neq \text{prerr} \wedge \text{pr_pos}(\text{pr}_1) = \text{pr_tomid},$
$\text{pr_exp}(\text{pr_at}(\text{pr_down}), \text{pr}_1) \leftrightarrow \text{pr}_1 \neq \text{prerr} \wedge \text{pr_pos}(\text{pr}_1) = \text{pr_todown},$

$\neg\ \text{pr_exp}(\text{pr_mvto}(\text{prpo}_1), \text{pr}_1),$
$\neg\ \text{pr_exp}(\text{pr_load}, \text{pr}_1),$
$\neg\ \text{pr_exp}(\text{pr_unload}, \text{pr}_1),$
$\neg\ \text{pr_exp}(\text{pr_at}(\text{pr_toup}), \text{pr}_1),$
$\neg\ \text{pr_exp}(\text{pr_at}(\text{pr_tomid}), \text{pr}_1),$
$\neg\ \text{pr_exp}(\text{pr_at}(\text{pr_todown}), \text{pr}_1),$

$\text{pr}_1 \neq \text{prerr} \wedge \text{pr_pos}(\text{pr}_1) = \text{pr_toup} \wedge$
$\text{pr_loaded}(\text{pr}_1) \wedge \text{pr}_2 = \text{pr_trans}(\text{pr_at}(\text{pr_up}), \text{pr}_1)$
$\rightarrow \text{pr}_2 \neq \text{prerr} \wedge \text{pr_pos}(\text{pr}_2) = \text{pr_up} \wedge \text{pr_loaded}(\text{pr}_2) \wedge \text{pr_pressed}(\text{pr}_2),$

$\text{pr}_1 = \text{prerr} \vee \text{pr_pos}(\text{pr}_1) \neq \text{pr_toup} \vee \neg\ \text{pr_loaded}(\text{pr}_1)$
$\rightarrow \text{pr_trans}(\text{pr_at}(\text{pr_up}), \text{pr}_1) = \text{prerr},$

$\text{pr}_1 \neq \text{prerr} \wedge \text{pr_pos}(\text{pr}_1) = \text{pr_todown} \wedge \text{pr}_2 = \text{pr_trans}(\text{pr_at}(\text{pr_down}), \text{pr}_1)$
$\rightarrow \text{pr}_2 \neq \text{prerr} \wedge \text{pr_pos}(\text{pr}_2) = \text{pr_down} \wedge$
$(\text{pr_loaded}(\text{pr}_2) \leftrightarrow \text{pr_loaded}(\text{pr}_1)) \wedge (\text{pr_pressed}(\text{pr}_2) \leftrightarrow \text{pr_pressed}(\text{pr}_1)),$

$\text{pr}_1 = \text{prerr} \vee \text{pr_pos}(\text{pr}_1) \neq \text{pr_todown}$
$\rightarrow \text{pr_trans}(\text{pr_at}(\text{pr_down}), \text{pr}_1) = \text{prerr},$

$\text{pr}_1 \neq \text{prerr} \wedge \text{pr_pos}(\text{pr}_1) = \text{pr_tomid} \wedge \text{pr}_2 = \text{pr_trans}(\text{pr_at}(\text{pr_mid}), \text{pr}_1)$
$\rightarrow \text{pr}_2 \neq \text{prerr} \wedge \text{pr_pos}(\text{pr}_2) = \text{pr_mid} \wedge$
$(\text{pr_loaded}(\text{pr}_2) \leftrightarrow \text{pr_loaded}(\text{pr}_1)) \wedge (\text{pr_pressed}(\text{pr}_2) \leftrightarrow \text{pr_pressed}(\text{pr}_1)),$

$$\begin{array}{l} \quad \mathrm{pr}_1 = \mathrm{prerr} \vee \mathrm{pr_pos}(\mathrm{pr}_1) \neq \mathrm{pr_tomid} \\ \to \mathrm{pr_trans}(\mathrm{pr_at}(\mathrm{pr_mid}), \mathrm{pr}_1) = \mathrm{prerr}, \end{array}$$

$$\begin{array}{l} \quad \mathrm{pr}_1 \neq \mathrm{prerr} \wedge (\mathrm{pr_pos}(\mathrm{pr}_1) = \mathrm{pr_mid} \vee \mathrm{pr_pos}(\mathrm{pr}_1) = \mathrm{pr_down}) \wedge \\ \quad \neg\, \mathrm{pr_loaded}(\mathrm{pr}_1) \wedge \mathrm{pr}_2 = \mathrm{pr_trans}(\mathrm{pr_load}, \mathrm{pr}_1) \\ \to \mathrm{pr}_2 \neq \mathrm{prerr} \wedge \mathrm{pr_pos}(\mathrm{pr}_2) = \mathrm{pr_mid} \wedge \mathrm{pr_loaded}(\mathrm{pr}_2) \wedge \neg\, \mathrm{pr_pressed}(\mathrm{pr}_2), \end{array}$$

$$\begin{array}{l} \quad \mathrm{pr}_1 = \mathrm{prerr} \vee \mathrm{pr_pos}(\mathrm{pr}_1) \neq \mathrm{pr_mid} \wedge \mathrm{pr_pos}(\mathrm{pr}_1) \neq \mathrm{pr_down} \vee \mathrm{pr_loaded}(\mathrm{pr}_1) \\ \to \mathrm{pr_trans}(\mathrm{pr_load}, \mathrm{pr}_1) = \mathrm{prerr}, \end{array}$$

$$\begin{array}{l} \quad \mathrm{pr}_1 \neq \mathrm{prerr} \wedge (\mathrm{pr_pos}(\mathrm{pr}_1) = \mathrm{pr_mid} \vee \mathrm{pr_pos}(\mathrm{pr}_1) = \mathrm{pr_down}) \wedge \\ \quad \mathrm{pr_loaded}(\mathrm{pr}_1) \wedge \mathrm{pr_pressed}(\mathrm{pr}_1) \wedge \mathrm{pr}_2 = \mathrm{pr_trans}(\mathrm{pr_unload}, \mathrm{pr}_1) \\ \to \mathrm{pr}_2 \neq \mathrm{prerr} \wedge \mathrm{pr_pos}(\mathrm{pr}_2) = \mathrm{pr_down} \wedge \neg\, \mathrm{pr_loaded}(\mathrm{pr}_2), \end{array}$$

$$\begin{array}{l} \quad \mathrm{pr}_1 = \mathrm{prerr} \vee \mathrm{pr_pos}(\mathrm{pr}_1) \neq \mathrm{pr_mid} \wedge \mathrm{pr_pos}(\mathrm{pr}_1) \neq \mathrm{pr_down} \vee \\ \quad \neg\, \mathrm{pr_loaded}(\mathrm{pr}_1) \vee \neg\, \mathrm{pr_pressed}(\mathrm{pr}_1) \\ \to \mathrm{pr_trans}(\mathrm{pr_unload}, \mathrm{pr}_1) = \mathrm{prerr}, \end{array}$$

$$\begin{array}{l} \quad \mathrm{pr}_1 \neq \mathrm{prerr} \wedge (\mathrm{pr_pos}(\mathrm{pr}_1) = \mathrm{pr_mid} \vee \mathrm{pr_pos}(\mathrm{pr}_1) = \mathrm{pr_down}) \wedge \\ \quad \mathrm{pr}_2 = \mathrm{pr_trans}(\mathrm{pr_mvto}(\mathrm{pr_up}), \mathrm{pr}_1) \\ \to \mathrm{pr}_2 \neq \mathrm{prerr} \wedge \mathrm{pr_pos}(\mathrm{pr}_2) = \mathrm{pr_toup} \wedge \\ \quad (\mathrm{pr_loaded}(\mathrm{pr}_2) \leftrightarrow \mathrm{pr_loaded}(\mathrm{pr}_1)) \wedge (\mathrm{pr_pressed}(\mathrm{pr}_2) \leftrightarrow \mathrm{pr_pressed}(\mathrm{pr}_1)), \end{array}$$

$$\begin{array}{l} \quad \mathrm{pr}_1 = \mathrm{prerr} \vee \mathrm{pr_pos}(\mathrm{pr}_1) \neq \mathrm{pr_down} \wedge \mathrm{pr_pos}(\mathrm{pr}_1) \neq \mathrm{pr_mid} \\ \to \mathrm{pr_trans}(\mathrm{pr_mvto}(\mathrm{pr_up}), \mathrm{pr}_1) = \mathrm{prerr}, \end{array}$$

$$\begin{array}{l} \quad \mathrm{pr}_1 \neq \mathrm{prerr} \wedge (\mathrm{pr_pos}(\mathrm{pr}_1) = \mathrm{pr_up} \vee \mathrm{pr_pos}(\mathrm{pr}_1) = \mathrm{pr_down}) \wedge \\ \quad \mathrm{pr}_2 = \mathrm{pr_trans}(\mathrm{pr_mvto}(\mathrm{pr_mid}), \mathrm{pr}_1) \\ \to \mathrm{pr}_2 \neq \mathrm{prerr} \wedge \mathrm{pr_pos}(\mathrm{pr}_2) = \mathrm{pr_tomid} \wedge \\ \quad (\mathrm{pr_loaded}(\mathrm{pr}_2) \leftrightarrow \mathrm{pr_loaded}(\mathrm{pr}_1)) \wedge (\mathrm{pr_pressed}(\mathrm{pr}_2) \leftrightarrow \mathrm{pr_pressed}(\mathrm{pr}_1)), \end{array}$$

$$\begin{array}{l} \quad \mathrm{pr}_1 = \mathrm{prerr} \vee \mathrm{pr_pos}(\mathrm{pr}_1) \neq \mathrm{pr_up} \wedge \mathrm{pr_pos}(\mathrm{pr}_1) \neq \mathrm{pr_down} \\ \to \mathrm{pr_trans}(\mathrm{pr_mvto}(\mathrm{pr_mid}), \mathrm{pr}_1) = \mathrm{prerr}, \end{array}$$

$$\begin{array}{l} \quad \mathrm{pr}_1 \neq \mathrm{prerr} \wedge (\mathrm{pr_pos}(\mathrm{pr}_1) = \mathrm{pr_up} \vee \mathrm{pr_pos}(\mathrm{pr}_1) = \mathrm{pr_mid}) \wedge \\ \quad \mathrm{pr}_2 = \mathrm{pr_trans}(\mathrm{pr_mvto}(\mathrm{pr_down}), \mathrm{pr}_1) \\ \to \mathrm{pr}_2 \neq \mathrm{prerr} \wedge \mathrm{pr_pos}(\mathrm{pr}_2) = \mathrm{pr_todown} \wedge \\ \quad (\mathrm{pr_loaded}(\mathrm{pr}_2) \leftrightarrow \mathrm{pr_loaded}(\mathrm{pr}_1)) \wedge (\mathrm{pr_pressed}(\mathrm{pr}_2) \leftrightarrow \mathrm{pr_pressed}(\mathrm{pr}_1)), \end{array}$$

$$\begin{array}{l} \quad \mathrm{pr}_1 = \mathrm{prerr} \vee \mathrm{pr_pos}(\mathrm{pr}_1) \neq \mathrm{pr_up} \wedge \mathrm{pr_pos}(\mathrm{pr}_1) \neq \mathrm{pr_mid} \\ \to \mathrm{pr_trans}(\mathrm{pr_mvto}(\mathrm{pr_down}), \mathrm{pr}_1) = \mathrm{prerr}, \end{array}$$

$$\begin{array}{l} \mathrm{pr_trans}(\mathrm{pr_mvto}(\mathrm{pr_toup}), \mathrm{pr}_1) = \mathrm{prerr}, \\ \mathrm{pr_trans}(\mathrm{pr_mvto}(\mathrm{pr_tomid}), \mathrm{pr}_1) = \mathrm{prerr}, \\ \mathrm{pr_trans}(\mathrm{pr_mvto}(\mathrm{pr_todown}), \mathrm{pr}_1) = \mathrm{prerr}, \\ \mathrm{pr_trans}(\mathrm{pr_at}(\mathrm{pr_toup}), \mathrm{pr}_1) = \mathrm{prerr}, \\ \mathrm{pr_trans}(\mathrm{pr_at}(\mathrm{pr_tomid}), \mathrm{pr}_1) = \mathrm{prerr}, \\ \mathrm{pr_trans}(\mathrm{pr_at}(\mathrm{pr_todown}), \mathrm{pr}_1) = \mathrm{prerr} \end{array}$$

end enrich

GENERIC_DBELT =
enrich DBELT_EVENTS **with**
sorts db_state;
constants dberr : db_state; dbinit : db_state;
functions db_trans : db_event $\times$ db_state $\rightarrow$ db_state ;
predicates

db_errp	:	db_state;
db_exp	:	db_event $\times$ db_state;
db_loaded	:	db_state;
db_pbinterrupted	:	db_state;
db_running	:	db_state;

variables db_2, db_1: db_state;
axioms
db_state **generated by** db_trans, dbinit;

db_errp(db_1) $\leftrightarrow$ db_1 = dberr,
dbinit $\neq$ dberr,
db_pbinterrupted(db_1) $\wedge$ $\neg$ db_loaded(db_1) $\rightarrow$ db_1 = dberr,

db_1 $\neq$ dberr
$\rightarrow$ (db_1 = db_2
$\leftrightarrow$ db_2 $\neq$ dberr
$\wedge$ (db_running(db_1) $\leftrightarrow$ db_running(db_2))
$\wedge$ (db_loaded(db_1) $\leftrightarrow$ db_loaded(db_2))
$\wedge$ (db_pbinterrupted(db_1) $\leftrightarrow$ db_pbinterrupted(db_2))),

db_exp(db_sheetfalls, db_1)
$\leftrightarrow$ db_1 $\neq$ dberr $\wedge$ db_running(db_1) $\wedge$ db_loaded(db_1) $\wedge$ db_pbinterrupted(db_1),
db_exp(db_pbinterruption, db_1)
$\leftrightarrow$ db_1 $\neq$ dberr $\wedge$ db_running(db_1) $\wedge$ db_loaded(db_1) $\wedge$ $\neg$ db_pbinterrupted(db_1),

$\neg$ db_exp(db_on, db_1),
$\neg$ db_exp(db_off, db_1),
$\neg$ db_exp(db_load, db_1),
$\neg$ db_exp(db_unload, db_1),

db_1 $\neq$ dberr $\wedge$ $\neg$ db_running(db_1) $\wedge$ db_2 = db_trans(db_on, db_1)
$\rightarrow$ db_2 $\neq$ dberr $\wedge$ db_running(db_2) $\wedge$
(db_loaded(db_2) $\leftrightarrow$ db_loaded(db_1)) $\wedge$
(db_pbinterrupted(db_2) $\leftrightarrow$ db_pbinterrupted(db_1)),

db_1 = dberr $\vee$ db_running(db_1)
$\rightarrow$ db_trans(db_on, db_1) = dberr,

db_1 $\neq$ dberr $\wedge$ db_running(db_1) $\wedge$ db_2 = db_trans(db_off, db_1)
$\rightarrow$ db_2 $\neq$ dberr $\wedge$ $\neg$ db_running(db_2) $\wedge$
(db_loaded(db_2) $\leftrightarrow$ db_loaded(db_1)) $\wedge$
(db_pbinterrupted(db_2) $\leftrightarrow$ db_pbinterrupted(db_1)),

db_1 = dberr $\vee$ $\neg$ db_running(db_1)
$\rightarrow$ db_trans(db_off, db_1) = dberr,

$db_1 \neq dberr \wedge db_running(db_1) \wedge$
$db_loaded(db_1) \wedge \neg\, db_pbinterrupted(db_1) \wedge db_2 = db_trans(db_pbinterruption, db_1)$
$\rightarrow db_2 \neq dberr \wedge db_running(db_2) \wedge db_loaded(db_2) \wedge db_pbinterrupted(db_2),$

$db_1 = dberr \vee \neg\, db_running(db_1) \vee \neg\, db_loaded(db_1) \vee db_pbinterrupted(db_1)$
$\rightarrow db_trans(db_pbinterruption, db_1) = dberr,$

$db_1 \neq dberr \wedge db_running(db_1) \wedge db_loaded(db_1) \wedge$
$db_pbinterrupted(db_1) \wedge db_2 = db_trans(db_sheetfalls, db_1)$
$\rightarrow db_2 \neq dberr \wedge db_running(db_2) \wedge \neg\, db_loaded(db_2) \wedge \neg\, db_pbinterrupted(db_2),$

$db_1 = dberr \vee \neg\, db_running(db_1) \vee \neg\, db_loaded(db_1) \vee \neg\, db_pbinterrupted(db_1)$
$\rightarrow db_trans(db_sheetfalls, db_1) = dberr,$

$db_1 \neq dberr \wedge \neg\, db_loaded(db_1) \wedge db_2 = db_trans(db_load, db_1)$
$\rightarrow db_2 \neq dberr \wedge (db_running(db_2) \leftrightarrow db_running(db_1)) \wedge$
$db_loaded(db_2) \wedge \neg\, db_pbinterrupted(db_2),$

$db_1 = dberr \vee db_loaded(db_1)$
$\rightarrow db_trans(db_load, db_1) = dberr,$

$db_1 \neq dberr \wedge db_loaded(db_1) \wedge$
$\neg\, db_running(db_1) \wedge db_pbinterrupted(db_1) \wedge db_2 = db_trans(db_unload, db_1)$
$\rightarrow db_2 \neq dberr \wedge \neg\, db_running(db_2) \wedge \neg\, db_loaded(db_2) \wedge \neg\, db_pbinterrupted(db_2),$

$db_1 = dberr \vee db_running(db_1) \vee \neg\, db_loaded(db_1) \vee \neg\, db_pbinterrupted(db_1)$
$\rightarrow db_trans(db_unload, db_1) = dberr$

end enrich

GENERIC_CRANE =
enrich CRANE_EVENTS **with**

sorts cr_state;
constants crerr : cr_state; crinit : cr_state;
functions

cr_apos	:	cr_state	→	cr_aposition	;
cr_pos	:	cr_state	→	cr_position	;
cr_trans	:	cr_event × cr_state	→	cr_state	;

predicates

cr_errp	:	cr_state;
cr_exp	:	cr_event × cr_state;
cr_aloaded	:	cr_state;

variables cre_1: cr_event; cr_3, cr_2, cr_1, cr_0: cr_state;
axioms

cr_state **generated by** cr_trans, crinit;

$cr_errp(cr_1) \leftrightarrow cr_1 = crerr,$
$crinit \neq crerr,$

$cr_apos(cr_1) \neq cr_aup \wedge (cr_pos(cr_1) = cr_tofb \vee cr_pos(cr_1) = cr_todb)$
$\rightarrow cr_1 = crerr,$

$$\begin{array}{l} \quad \mathrm{cr}_1 \neq \mathrm{crerr} \\ \rightarrow \quad (\mathrm{cr}_1 = \mathrm{cr}_2 \\ \quad \leftrightarrow \quad \mathrm{cr}_2 \neq \mathrm{crerr} \\ \qquad \wedge\ \mathrm{cr_pos}(\mathrm{cr}_1) = \mathrm{cr_pos}(\mathrm{cr}_2) \\ \qquad \wedge\ \mathrm{cr_apos}(\mathrm{cr}_1) = \mathrm{cr_apos}(\mathrm{cr}_2) \\ \qquad \wedge\ (\mathrm{cr_aloaded}(\mathrm{cr}_1) \leftrightarrow \mathrm{cr_aloaded}(\mathrm{cr}_2))\ ), \end{array}$$

$$\begin{array}{l} \mathrm{cr_exp}(\mathrm{cr_aat}(\mathrm{cr_aup}), \mathrm{cr}_1) \leftrightarrow \mathrm{cr}_1 \neq \mathrm{crerr} \wedge \mathrm{cr_apos}(\mathrm{cr}_1) = \mathrm{cr_atoup}, \\ \mathrm{cr_exp}(\mathrm{cr_aat}(\mathrm{cr_amid}), \mathrm{cr}_1) \leftrightarrow \mathrm{cr}_1 \neq \mathrm{crerr} \wedge \mathrm{cr_apos}(\mathrm{cr}_1) = \mathrm{cr_atomid}, \\ \mathrm{cr_exp}(\mathrm{cr_aat}(\mathrm{cr_adown}), \mathrm{cr}_1) \leftrightarrow \mathrm{cr}_1 \neq \mathrm{crerr} \wedge \mathrm{cr_apos}(\mathrm{cr}_1) = \mathrm{cr_atodown}, \\ \mathrm{cr_exp}(\mathrm{cr_at}(\mathrm{cr_db}), \mathrm{cr}_1) \leftrightarrow \mathrm{cr}_1 \neq \mathrm{crerr} \wedge \mathrm{cr_pos}(\mathrm{cr}_1) = \mathrm{cr_todb}, \\ \mathrm{cr_exp}(\mathrm{cr_at}(\mathrm{cr_fb}), \mathrm{cr}_1) \leftrightarrow \mathrm{cr}_1 \neq \mathrm{crerr} \wedge \mathrm{cr_pos}(\mathrm{cr}_1) = \mathrm{cr_tofb}, \end{array}$$

$$\begin{array}{l} \neg\ \mathrm{cr_exp}(\mathrm{cr_mvto}(\mathrm{crpo}_1), \mathrm{cr}_1), \\ \neg\ \mathrm{cr_exp}(\mathrm{cr_amvto}(\mathrm{crap}_1), \mathrm{cr}_1), \\ \neg\ \mathrm{cr_exp}(\mathrm{cr_load}, \mathrm{cr}_1), \\ \neg\ \mathrm{cr_exp}(\mathrm{cr_unload}, \mathrm{cr}_1), \\ \neg\ \mathrm{cr_exp}(\mathrm{cr_aat}(\mathrm{cr_atoup}), \mathrm{cr}_1), \\ \neg\ \mathrm{cr_exp}(\mathrm{cr_aat}(\mathrm{cr_atomid}), \mathrm{cr}_1), \\ \neg\ \mathrm{cr_exp}(\mathrm{cr_aat}(\mathrm{cr_atodown}), \mathrm{cr}_1), \end{array}$$

$$\begin{array}{l} \quad \mathrm{cr}_1 \neq \mathrm{crerr} \wedge \mathrm{cr_apos}(\mathrm{cr}_1) = \mathrm{cr_aup} \wedge \mathrm{cr_pos}(\mathrm{cr}_1) = \mathrm{cr_db}\ \wedge \\ \quad \mathrm{cr}_2 = \mathrm{cr_trans}(\mathrm{cr_mvto}(\mathrm{cr_fb}), \mathrm{cr}_1) \\ \rightarrow \mathrm{cr}_2 \neq \mathrm{crerr} \wedge \mathrm{cr_apos}(\mathrm{cr}_2) = \mathrm{cr_aup}\ \wedge \\ \quad \mathrm{cr_pos}(\mathrm{cr}_2) = \mathrm{cr_tofb} \wedge (\mathrm{cr_aloaded}(\mathrm{cr}_2) \leftrightarrow \mathrm{cr_aloaded}(\mathrm{cr}_1)), \end{array}$$

$$\begin{array}{l} \quad \mathrm{cr}_1 = \mathrm{crerr} \vee \mathrm{cr_apos}(\mathrm{cr}_1) \neq \mathrm{cr_aup} \vee \mathrm{cr_pos}(\mathrm{cr}_1) \neq \mathrm{cr_db} \\ \rightarrow \mathrm{cr_trans}(\mathrm{cr_mvto}(\mathrm{cr_fb}), \mathrm{cr}_1) = \mathrm{crerr}, \end{array}$$

$$\begin{array}{l} \quad \mathrm{cr}_1 \neq \mathrm{crerr} \wedge \mathrm{cr_apos}(\mathrm{cr}_1) = \mathrm{cr_aup} \wedge \mathrm{cr_pos}(\mathrm{cr}_1) = \mathrm{cr_fb}\ \wedge \\ \quad \mathrm{cr}_2 = \mathrm{cr_trans}(\mathrm{cr_mvto}(\mathrm{cr_db}), \mathrm{cr}_1) \\ \rightarrow \mathrm{cr}_2 \neq \mathrm{crerr} \wedge \mathrm{cr_apos}(\mathrm{cr}_2) = \mathrm{cr_aup} \wedge \mathrm{cr_pos}(\mathrm{cr}_2) = \mathrm{cr_todb}\ \wedge \\ \quad (\mathrm{cr_aloaded}(\mathrm{cr}_2) \leftrightarrow \mathrm{cr_aloaded}(\mathrm{cr}_1)), \end{array}$$

$$\begin{array}{l} \quad \mathrm{cr}_1 = \mathrm{crerr} \vee \mathrm{cr_apos}(\mathrm{cr}_1) \neq \mathrm{cr_aup} \vee \mathrm{cr_pos}(\mathrm{cr}_1) \neq \mathrm{cr_fb} \\ \rightarrow \mathrm{cr_trans}(\mathrm{cr_mvto}(\mathrm{cr_db}), \mathrm{cr}_1) = \mathrm{crerr}, \end{array}$$

$$\begin{array}{l} \quad \mathrm{cr}_1 \neq \mathrm{crerr} \wedge \mathrm{cr_apos}(\mathrm{cr}_1) = \mathrm{cr_aup}\ \wedge \\ \quad \mathrm{cr_pos}(\mathrm{cr}_1) = \mathrm{cr_tofb} \wedge \mathrm{cr}_2 = \mathrm{cr_trans}(\mathrm{cr_at}(\mathrm{cr_fb}), \mathrm{cr}_1) \\ \rightarrow \mathrm{cr}_2 \neq \mathrm{crerr} \wedge \mathrm{cr_apos}(\mathrm{cr}_2) = \mathrm{cr_aup}\ \wedge \\ \quad \mathrm{cr_pos}(\mathrm{cr}_2) = \mathrm{cr_fb} \wedge (\mathrm{cr_aloaded}(\mathrm{cr}_2) \leftrightarrow \mathrm{cr_aloaded}(\mathrm{cr}_1)), \end{array}$$

$$\begin{array}{l} \quad \mathrm{cr}_1 = \mathrm{crerr} \vee \mathrm{cr_apos}(\mathrm{cr}_1) \neq \mathrm{cr_aup} \vee \mathrm{cr_pos}(\mathrm{cr}_1) \neq \mathrm{cr_tofb} \\ \rightarrow \mathrm{cr_trans}(\mathrm{cr_at}(\mathrm{cr_fb}), \mathrm{cr}_1) = \mathrm{crerr}, \end{array}$$

$$\begin{aligned} & \mathrm{cr}_1 \neq \mathrm{crerr} \wedge \mathrm{cr_apos}(\mathrm{cr}_1) = \mathrm{cr_aup} \wedge \\ & \mathrm{cr_pos}(\mathrm{cr}_1) = \mathrm{cr_todb} \wedge \mathrm{cr}_2 = \mathrm{cr_trans}(\mathrm{cr_at}(\mathrm{cr_db}), \mathrm{cr}_1) \\ \rightarrow\ & \mathrm{cr}_2 \neq \mathrm{crerr} \wedge \mathrm{cr_apos}(\mathrm{cr}_2) = \mathrm{cr_aup} \wedge \\ & \mathrm{cr_pos}(\mathrm{cr}_2) = \mathrm{cr_db} \wedge (\mathrm{cr_aloaded}(\mathrm{cr}_2) \leftrightarrow \mathrm{cr_aloaded}(\mathrm{cr}_1)), \end{aligned}$$

$$\begin{aligned} & \mathrm{cr}_1 = \mathrm{crerr} \vee \mathrm{cr_apos}(\mathrm{cr}_1) \neq \mathrm{cr_aup} \vee \mathrm{cr_pos}(\mathrm{cr}_1) \neq \mathrm{cr_todb} \\ \rightarrow\ & \mathrm{cr_trans}(\mathrm{cr_at}(\mathrm{cr_fb}), \mathrm{cr}_1) = \mathrm{crerr}, \end{aligned}$$

$$\begin{aligned} & \mathrm{cr}_1 \neq \mathrm{crerr} \wedge (\mathrm{cr_apos}(\mathrm{cr}_1) = \mathrm{cr_adown} \vee \mathrm{cr_apos}(\mathrm{cr}_1) = \mathrm{cr_amid}) \wedge \\ & (\mathrm{cr_pos}(\mathrm{cr}_1) = \mathrm{cr_fb} \vee \mathrm{cr_pos}(\mathrm{cr}_1) = \mathrm{cr_db}) \wedge \mathrm{cr}_2 = \mathrm{cr_trans}(\mathrm{cr_amvto}(\mathrm{cr_aup}), \mathrm{cr}_1) \\ \rightarrow\ & \mathrm{cr}_2 \neq \mathrm{crerr} \wedge \mathrm{cr_pos}(\mathrm{cr}_2) = \mathrm{cr_pos}(\mathrm{cr}_1) \wedge \\ & \mathrm{cr_apos}(\mathrm{cr}_2) = \mathrm{cr_atoup} \wedge (\mathrm{cr_aloaded}(\mathrm{cr}_2) \leftrightarrow \mathrm{cr_aloaded}(\mathrm{cr}_1)), \end{aligned}$$

$$\begin{aligned} & \mathrm{cr}_1 = \mathrm{crerr} \vee \mathrm{cr_apos}(\mathrm{cr}_1) \neq \mathrm{cr_adown} \wedge \mathrm{cr_apos}(\mathrm{cr}_1) \neq \mathrm{cr_amid} \vee \\ & \mathrm{cr_pos}(\mathrm{cr}_1) \neq \mathrm{cr_fb} \wedge \mathrm{cr_pos}(\mathrm{cr}_1) \neq \mathrm{cr_db} \\ \rightarrow\ & \mathrm{cr_trans}(\mathrm{cr_amvto}(\mathrm{cr_aup}), \mathrm{cr}_1) = \mathrm{crerr}, \end{aligned}$$

$$\begin{aligned} & \mathrm{cr}_1 \neq \mathrm{crerr} \wedge (\mathrm{cr_apos}(\mathrm{cr}_1) = \mathrm{cr_adown} \vee \mathrm{cr_apos}(\mathrm{cr}_1) = \mathrm{cr_aup}) \wedge \\ & (\mathrm{cr_pos}(\mathrm{cr}_1) = \mathrm{cr_fb} \vee \mathrm{cr_pos}(\mathrm{cr}_1) = \mathrm{cr_db}) \wedge \mathrm{cr}_2 = \mathrm{cr_trans}(\mathrm{cr_amvto}(\mathrm{cr_amid}), \mathrm{cr}_1) \\ \rightarrow\ & \mathrm{cr}_2 \neq \mathrm{crerr} \wedge \mathrm{cr_pos}(\mathrm{cr}_2) = \mathrm{cr_pos}(\mathrm{cr}_1) \wedge \\ & \mathrm{cr_apos}(\mathrm{cr}_2) = \mathrm{cr_atomid} \wedge (\mathrm{cr_aloaded}(\mathrm{cr}_2) \leftrightarrow \mathrm{cr_aloaded}(\mathrm{cr}_1)), \end{aligned}$$

$$\begin{aligned} & \mathrm{cr}_1 = \mathrm{crerr} \vee \mathrm{cr_apos}(\mathrm{cr}_1) \neq \mathrm{cr_adown} \wedge \mathrm{cr_apos}(\mathrm{cr}_1) \neq \mathrm{cr_aup} \vee \\ & \mathrm{cr_pos}(\mathrm{cr}_1) \neq \mathrm{cr_fb} \wedge \mathrm{cr_pos}(\mathrm{cr}_1) \neq \mathrm{cr_db} \\ \rightarrow\ & \mathrm{cr_trans}(\mathrm{cr_amvto}(\mathrm{cr_amid}), \mathrm{cr}_1) = \mathrm{crerr}, \end{aligned}$$

$$\begin{aligned} & \mathrm{cr}_1 \neq \mathrm{crerr} \wedge (\mathrm{cr_apos}(\mathrm{cr}_1) = \mathrm{cr_amid} \vee \mathrm{cr_apos}(\mathrm{cr}_1) = \mathrm{cr_aup}) \wedge \\ & (\mathrm{cr_pos}(\mathrm{cr}_1) = \mathrm{cr_fb} \vee \mathrm{cr_pos}(\mathrm{cr}_1) = \mathrm{cr_db}) \wedge \mathrm{cr}_2 = \mathrm{cr_trans}(\mathrm{cr_amvto}(\mathrm{cr_adown}), \mathrm{cr}_1) \\ \rightarrow\ & \mathrm{cr}_2 \neq \mathrm{crerr} \wedge \mathrm{cr_pos}(\mathrm{cr}_2) = \mathrm{cr_pos}(\mathrm{cr}_1) \wedge \\ & \mathrm{cr_apos}(\mathrm{cr}_2) = \mathrm{cr_atodown} \wedge (\mathrm{cr_aloaded}(\mathrm{cr}_2) \leftrightarrow \mathrm{cr_aloaded}(\mathrm{cr}_1)), \end{aligned}$$

$$\begin{aligned} & \mathrm{cr}_1 = \mathrm{crerr} \vee \mathrm{cr_apos}(\mathrm{cr}_1) \neq \mathrm{cr_amid} \wedge \\ & \mathrm{cr_apos}(\mathrm{cr}_1) \neq \mathrm{cr_aup} \vee \mathrm{cr_pos}(\mathrm{cr}_1) \neq \mathrm{cr_fb} \wedge \mathrm{cr_pos}(\mathrm{cr}_1) \neq \mathrm{cr_db} \\ \rightarrow\ & \mathrm{cr_trans}(\mathrm{cr_amvto}(\mathrm{cr_adown}), \mathrm{cr}_1) = \mathrm{crerr}, \end{aligned}$$

$$\begin{aligned} & \mathrm{cr}_1 \neq \mathrm{crerr} \wedge \mathrm{cr_apos}(\mathrm{cr}_1) = \mathrm{cr_atoup} \wedge \\ & (\mathrm{cr_pos}(\mathrm{cr}_1) = \mathrm{cr_fb} \vee \mathrm{cr_pos}(\mathrm{cr}_1) = \mathrm{cr_db}) \wedge \mathrm{cr}_2 = \mathrm{cr_trans}(\mathrm{cr_aat}(\mathrm{cr_aup}), \mathrm{cr}_1) \\ \rightarrow\ & \mathrm{cr}_2 \neq \mathrm{crerr} \wedge \mathrm{cr_pos}(\mathrm{cr}_2) = \mathrm{cr_pos}(\mathrm{cr}_1) \wedge \\ & \mathrm{cr_apos}(\mathrm{cr}_2) = \mathrm{cr_aup} \wedge (\mathrm{cr_aloaded}(\mathrm{cr}_2) \leftrightarrow \mathrm{cr_aloaded}(\mathrm{cr}_1)), \end{aligned}$$

$$\begin{aligned} & \mathrm{cr}_1 = \mathrm{crerr} \vee \mathrm{cr_apos}(\mathrm{cr}_1) \neq \mathrm{cr_atoup} \vee \mathrm{cr_pos}(\mathrm{cr}_1) \neq \mathrm{cr_fb} \wedge \mathrm{cr_pos}(\mathrm{cr}_1) \neq \mathrm{cr_db} \\ \rightarrow\ & \mathrm{cr_trans}(\mathrm{cr_aat}(\mathrm{cr_aup}), \mathrm{cr}_1) = \mathrm{crerr}, \end{aligned}$$

$$\begin{aligned} & \mathrm{cr}_1 \neq \mathrm{crerr} \wedge \mathrm{cr_apos}(\mathrm{cr}_1) = \mathrm{cr_atomid} \wedge \\ & (\mathrm{cr_pos}(\mathrm{cr}_1) = \mathrm{cr_fb} \vee \mathrm{cr_pos}(\mathrm{cr}_1) = \mathrm{cr_db}) \wedge \mathrm{cr}_2 = \mathrm{cr_trans}(\mathrm{cr_aat}(\mathrm{cr_amid}), \mathrm{cr}_1) \\ \rightarrow\ & \mathrm{cr}_2 \neq \mathrm{crerr} \wedge \mathrm{cr_pos}(\mathrm{cr}_2) = \mathrm{cr_pos}(\mathrm{cr}_1) \wedge \\ & \mathrm{cr_apos}(\mathrm{cr}_2) = \mathrm{cr_amid} \wedge (\mathrm{cr_aloaded}(\mathrm{cr}_2) \leftrightarrow \mathrm{cr_aloaded}(\mathrm{cr}_1)), \end{aligned}$$

$$\begin{aligned} & \mathrm{cr}_1 = \mathrm{crerr} \vee \mathrm{cr_apos}(\mathrm{cr}_1) \neq \mathrm{cr_atomid} \vee \mathrm{cr_pos}(\mathrm{cr}_1) \neq \mathrm{cr_fb} \wedge \mathrm{cr_pos}(\mathrm{cr}_1) \neq \mathrm{cr_db} \\ \rightarrow\ & \mathrm{cr_trans}(\mathrm{cr_aat}(\mathrm{cr_amid}), \mathrm{cr}_1) = \mathrm{crerr}, \end{aligned}$$

$cr_1 \neq crerr \wedge cr_apos(cr_1) = cr_atodown \wedge$
$(cr_pos(cr_1) = cr_fb \vee cr_pos(cr_1) = cr_db) \wedge cr_2 = cr_trans(cr_aat(cr_adown), cr_1)$
$\rightarrow cr_2 \neq crerr \wedge cr_pos(cr_2) = cr_pos(cr_1) \wedge$
$cr_apos(cr_2) = cr_adown \wedge (cr_aloaded(cr_2) \leftrightarrow cr_aloaded(cr_1)),$

$cr_1 = crerr \vee cr_apos(cr_1) \neq cr_atodown \vee cr_pos(cr_1) \neq cr_fb \wedge cr_pos(cr_1) \neq cr_db$
$\rightarrow cr_trans(cr_aat(cr_adown), cr_1) = crerr,$

$cr_1 \neq crerr \wedge (cr_apos(cr_1) = cr_adown \vee cr_apos(cr_1) = cr_amid) \wedge$
$\neg\, cr_aloaded(cr_1) \wedge cr_2 = cr_trans(cr_load, cr_1)$
$\rightarrow cr_2 \neq crerr \wedge cr_aloaded(cr_2) \wedge$
$cr_pos(cr_2) = cr_pos(cr_1) \wedge cr_apos(cr_2) = cr_apos(cr_1),$

$cr_1 = crerr \vee cr_apos(cr_1) \neq cr_adown \wedge$
$cr_apos(cr_1) \neq cr_amid \vee cr_aloaded(cr_1)$
$\rightarrow cr_trans(cr_load, cr_1) = crerr,$

$cr_1 \neq crerr \wedge (cr_apos(cr_1) = cr_adown \vee cr_apos(cr_1) = cr_amid) \wedge$
$cr_aloaded(cr_1) \wedge cr_2 = cr_trans(cr_unload, cr_1)$
$\rightarrow cr_2 \neq crerr \wedge \neg\, cr_aloaded(cr_2) \wedge$
$cr_pos(cr_2) = cr_pos(cr_1) \wedge cr_apos(cr_2) = cr_apos(cr_1),$

$cr_1 = crerr \vee cr_apos(cr_1) \neq cr_adown \wedge$
$cr_apos(cr_1) \neq cr_amid \vee \neg\, cr_aloaded(cr_1)$
$\rightarrow cr_trans(cr_unload, cr_1) = crerr$

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
$fbe_1 \neq fb_unload \rightarrow fb_tr(fbe_1, fb_1) = fb_trans(fbe_1, fb_1),$
$fb_tr(fb_unload, fb_1) = fberr$
end enrich

TABLE =
enrich GENERIC_TABLE **with**
functions ta_tr : ta_event $\times$ ta_state $\rightarrow$ ta_state ;
axioms
$hte_1 \neq ta_load \wedge hte_1 \neq ta_unload \rightarrow ta_tr(hte_1, ta_1) = ta_trans(hte_1, ta_1),$
$ta_pos(ta_1) \neq ta_down \rightarrow ta_tr(ta_load, ta_1) = taerr,$
$ta_pos(ta_1) = ta_down \rightarrow ta_tr(ta_load, ta_1) = ta_trans(ta_load, ta_1),$
$ta_tpos(ta_1) \neq ta_ro \rightarrow ta_tr(ta_unload, ta_1) = taerr,$
$ta_tpos(ta_1) = ta_ro \rightarrow ta_tr(ta_unload, ta_1) = ta_trans(ta_unload, ta_1)$
end enrich

ROBOT =
enrich GENERIC_ROBOT **with**
functions ro_tr : ro_event $\times$ ro_state $\rightarrow$ ro_state ;
axioms

ro_tr(ro_at(ropo$_1$), ro$_1$) = ro_trans(ro_at(ropo$_1$), ro$_1$),
ro_tr(ro_mvto(ropo$_1$), ro$_1$) = ro_trans(ro_mvto(ropo$_1$), ro$_1$),

ro_pos(ro$_1$) = ro_a1ta $\wedge$ ro_a1pos(ro$_1$) = ro_amid
$\rightarrow$ ro_tr(ro_a1load, ro$_1$) = ro_trans(ro_a1load, ro$_1$),

ro_pos(ro$_1$) $\neq$ ro_a1ta $\vee$ ro_a1pos(ro$_1$) $\neq$ ro_amid
$\rightarrow$ ro_tr(ro_a1load, ro$_1$) = roerr,

ro_pos(ro$_1$) = ro_a1pr $\wedge$ ro_a1pos(ro$_1$) = ro_aout
$\rightarrow$ ro_tr(ro_a1unload, ro$_1$) = ro_trans(ro_a1unload, ro$_1$),

ro_pos(ro$_1$) $\neq$ ro_a1pr $\vee$ ro_a1pos(ro$_1$) $\neq$ ro_aout
$\rightarrow$ ro_tr(ro_a1unload, ro$_1$) = ro_trans(ro_a1unload, ro$_1$),

ro_pos(ro$_1$) = ro_a2pr $\wedge$ ro_a2pos(ro$_1$) = ro_aout
$\rightarrow$ ro_tr(ro_a2load, ro$_1$) = ro_trans(ro_a2load, ro$_1$),

ro_pos(ro$_1$) $\neq$ ro_a2pr $\vee$ ro_a2pos(ro$_1$) $\neq$ ro_aout
$\rightarrow$ ro_tr(ro_a2load, ro$_1$) = roerr,

ro_pos(ro$_1$) = ro_a2db $\wedge$ ro_a2pos(ro$_1$) = ro_amid
$\rightarrow$ ro_tr(ro_a2unload, ro$_1$) = ro_trans(ro_a2load, ro$_1$),

ro_pos(ro$_1$) $\neq$ ro_a2db $\vee$ ro_a2pos(ro$_1$) $\neq$ ro_amid
$\rightarrow$ ro_tr(ro_a2unload, ro$_1$) = roerr,

ro_tr(ro_a1at(roap$_1$), ro$_1$) = ro_trans(ro_a1at(roap$_1$), ro$_1$),
ro_tr(ro_a1mv(roap$_1$), ro$_1$) = ro_trans(ro_a1mv(roap$_1$), ro$_1$),
ro_tr(ro_a2at(roap$_1$), ro$_1$) = ro_trans(ro_a2at(roap$_1$), ro$_1$),
ro_tr(ro_a2mv(roap$_1$), ro$_1$) = ro_trans(ro_a2mv(roap$_1$), ro$_1$)

end enrich

PRESS =
enrich GENERIC_PRESS **with**
functions pr_tr : pr_event $\times$ pr_state $\rightarrow$ pr_state ;
axioms

pr_tr(pr_at(prpo$_1$), pr$_1$) = pr_trans(pr_at(prpo$_1$), pr$_1$),
pr_tr(pr_mvto(prpo$_1$), pr$_1$) = pr_trans(pr_mvto(prpo$_1$), pr$_1$),
pr_pos(pr$_1$) = pr_mid $\rightarrow$ pr_tr(pr_load, pr$_1$) = pr_trans(pr_load, pr$_1$),
pr_pos(pr$_1$) $\neq$ pr_mid $\rightarrow$ pr_tr(pr_load, pr$_1$) = prerr,
pr_pos(pr$_1$) = pr_down $\rightarrow$ pr_tr(pr_unload, pr$_1$) = pr_trans(pr_unload, pr$_1$),
pr_pos(pr$_1$) $\neq$ pr_down $\rightarrow$ pr_tr(pr_unload, pr$_1$) = prerr

end enrich

DBELT =
enrich GENERIC_DBELT **with**
functions db_tr : db_event × db_state → db_state ;
axioms
$dbe_1 \neq$ db_sheetfalls → db_tr(dbe_1, db_1) = db_trans(dbe_1, db_1),
db_tr(db_sheetfalls, db_1) = dberr
end enrich

CRANE =
enrich GENERIC_CRANE **with**
functions cr_tr : cr_event × cr_state → cr_state ;
axioms
cr_tr(cr_mvto($crpo_1$), cr_1) = cr_trans(cr_mvto($crpo_1$), cr_1),

$crap_1 \neq$ cr_adown ∨ cr_pos(cr_1) ≠ cr_fb
→ cr_tr(cr_amvto($crap_1$), cr_1) = cr_trans(cr_amvto($crap_1$), cr_1),

cr_pos(cr_1) = cr_fb
→ cr_tr(cr_amvto(cr_adown), cr_1) = crerr,

$crap_1 \neq$ cr_adown ∨ cr_pos(cr_1) ≠ cr_fb
→ cr_tr(cr_aat($crap_1$), cr_1) = cr_trans(cr_aat($crap_1$), cr_1),

cr_pos(cr_1) = cr_fb → cr_tr(cr_aat(cr_adown), cr_1) = crerr,
cr_pos(cr_1) = cr_db ∧ cr_apos(cr_1) = cr_adown
→ cr_tr(cr_load, cr_1) = cr_trans(cr_load, cr_1),

cr_pos(cr_1) ≠ cr_db ∨ cr_apos(cr_1) ≠ cr_adown
→ cr_tr(cr_load, cr_1) = crerr,

cr_pos(cr_1) = cr_fb ∧ cr_apos(cr_1) = cr_amid
→ cr_tr(cr_unload, cr_1) = cr_trans(cr_unload, cr_1),

cr_pos(cr_1) ≠ cr_fb ∨ cr_apos(cr_1) ≠ cr_amid
→ cr_tr(cr_unload, cr_1) = 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	$\rightarrow$	elem	;
cons	:	elem $\times$ list	$\rightarrow$	list	;
cdr	:	list	$\rightarrow$	list	;
append	:	list $\times$ list	$\rightarrow$	list	;
reverse	:	list	$\rightarrow$	list	;

variables l_1, l: list;
axioms
list **generated by** nil, cons;
car(cons(a, l)) = a,
cdr(cons(a, l)) = l,
nil $\neq$ cons(a, l),
l = nil $\vee$ l = cons(car(l), cdr(l)),
append(nil, l) = l,
append(cons(a, l_1), l) = cons(a, append(l_1, 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 $\times$ ta_state $\times$ ro_state $\times$ pr_state $\times$ db_state $\times$ cr_state	$\rightarrow$	state	;
trans	:	event $\times$ state	$\rightarrow$	state	;
fbc	:	state	$\rightarrow$	fb_state	;
tac	:	state	$\rightarrow$	ta_state	;
roc	:	state	$\rightarrow$	ro_state	;
prc	:	state	$\rightarrow$	pr_state	;
dbc	:	state	$\rightarrow$	db_state	;
crc	:	state	$\rightarrow$	cr_state	;
cfb	:	fb_ev	$\rightarrow$	event	;
cht	:	ta_ev	$\rightarrow$	event	;
cro	:	ro_ev	$\rightarrow$	event	;
cpr	:	pr_ev	$\rightarrow$	event	;
cdb	:	db_ev	$\rightarrow$	event	;
ccr	:	cr_ev	$\rightarrow$	event	;

predicates

safe	:	state;
notshared	:	event;
expe	:	event $\times$ state;

variables e_0, e: event; fz_2, fz_1, s_2, s_1, s: state;
fbe_2: fb_ev; hte_2: ta_ev; roe_2: ro_ev; cre_2: cr_ev; pre_2: pr_ev; dbe_2: db_ev;

axioms

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

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

$\text{safe}(\text{mkstate}(\text{fb}_1, \text{ta}_1, \text{ro}_1, \text{pr}_1, \text{db}_1, \text{cr}_1))$
$\leftrightarrow \text{fb}_1 \neq \text{fberr} \wedge \text{ta}_1 \neq \text{taerr} \wedge \text{ro}_1 \neq \text{roerr} \wedge$
$\text{pr}_1 \neq \text{prerr} \wedge \text{db}_1 \neq \text{dberr} \wedge \text{cr}_1 \neq \text{crerr} \wedge$
$\neg (\text{ro_pos}(\text{ro}_1) = \text{ro_a1pr} \wedge \text{ro_a1pos}(\text{ro}_1) \neq \text{ro_ain}) \wedge$
$\neg (\text{ro_pos}(\text{ro}_1) = \text{ro_a2pr} \wedge \text{ro_a2pos}(\text{ro}_1) \neq \text{ro_ain})$,

$\text{fbc}(\text{mkstate}(\text{fb}_1, \text{ta}_1, \text{ro}_1, \text{pr}_1, \text{db}_1, \text{cr}_1)) = \text{fb}_1$,
$\text{tac}(\text{mkstate}(\text{fb}_1, \text{ta}_1, \text{ro}_1, \text{pr}_1, \text{db}_1, \text{cr}_1)) = \text{ta}_1$,
$\text{roc}(\text{mkstate}(\text{fb}_1, \text{ta}_1, \text{ro}_1, \text{pr}_1, \text{db}_1, \text{cr}_1)) = \text{ro}_1$,
$\text{prc}(\text{mkstate}(\text{fb}_1, \text{ta}_1, \text{ro}_1, \text{pr}_1, \text{db}_1, \text{cr}_1)) = \text{pr}_1$,
$\text{dbc}(\text{mkstate}(\text{fb}_1, \text{ta}_1, \text{ro}_1, \text{pr}_1, \text{db}_1, \text{cr}_1)) = \text{db}_1$,
$\text{crc}(\text{mkstate}(\text{fb}_1, \text{ta}_1, \text{ro}_1, \text{pr}_1, \text{db}_1, \text{cr}_1)) = \text{cr}_1$,

$\text{s}_1 = \text{s}_2$
$\leftrightarrow \text{fbc}(\text{s}_1) = \text{fbc}(\text{s}_2) \wedge \text{tac}(\text{s}_1) = \text{tac}(\text{s}_2) \wedge \text{roc}(\text{s}_1) = \text{roc}(\text{s}_2) \wedge$
$\text{prc}(\text{s}_1) = \text{prc}(\text{s}_2) \wedge \text{dbc}(\text{s}_1) = \text{dbc}(\text{s}_2) \wedge \text{crc}(\text{s}_1) = \text{crc}(\text{s}_2)$,

$\forall \text{e}. (\exists \text{fbe}_1.\text{e} = \text{cfb}(\text{fbe}_1)) \vee (\exists \text{hte}_1.\text{e} = \text{cht}(\text{hte}_1)) \vee (\exists \text{roe}_1.\text{e} = \text{cro}(\text{roe}_1)) \vee$
$(\exists \text{pre}_1.\text{e} = \text{cpr}(\text{pre}_1)) \vee (\exists \text{dbe}_1.\text{e} = \text{cdb}(\text{dbe}_1)) \vee (\exists \text{cre}_1.\text{e} = \text{ccr}(\text{cre}_1))$,

$\text{cfb}(\text{fbe}_1) = \text{cfb}(\text{fbe}_2) \leftrightarrow \text{fbe}_1 = \text{fbe}_2$,
$\text{cht}(\text{hte}_1) = \text{cht}(\text{hte}_2) \leftrightarrow \text{hte}_1 = \text{hte}_2$,
$\text{cro}(\text{roe}_1) = \text{cro}(\text{roe}_2) \leftrightarrow \text{roe}_1 = \text{roe}_2$,
$\text{cpr}(\text{pre}_1) = \text{cpr}(\text{pre}_2) \leftrightarrow \text{pre}_1 = \text{pre}_2$,
$\text{cdb}(\text{dbe}_1) = \text{cdb}(\text{dbe}_2) \leftrightarrow \text{dbe}_1 = \text{dbe}_2$,
$\text{ccr}(\text{cre}_1) = \text{ccr}(\text{cre}_2) \leftrightarrow \text{cre}_1 = \text{cre}_2$,

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),

$\text{fbe}_1 \neq \text{fb_unload} \wedge \text{fbe}_1 \neq \text{fb_sheetfalls} \rightarrow \text{notshared}(\text{cfb}(\text{fbe}_1))$,
$\text{hte}_1 \neq \text{ta_load} \wedge \text{hte}_1 \neq \text{ta_unload} \rightarrow \text{notshared}(\text{cht}(\text{hte}_1))$,
$\text{roe}_1 \neq \text{ro_a1load} \wedge \text{roe}_1 \neq \text{ro_a1unload} \wedge \text{roe}_1 \neq \text{ro_a2load} \wedge \text{roe}_1 \neq \text{ro_a2unload}$
$\rightarrow \text{notshared}(\text{cro}(\text{roe}_1))$,
$\text{pre}_1 \neq \text{pr_load} \wedge \text{pre}_1 \neq \text{pr_unload} \rightarrow \text{notshared}(\text{cpr}(\text{pre}_1))$,
$\text{dbe}_1 \neq \text{db_load} \wedge \text{dbe}_1 \neq \text{db_load} \rightarrow \text{notshared}(\text{cdb}(\text{dbe}_1))$,
$\text{cre}_1 \neq \text{cr_load} \wedge \text{cre}_1 \neq \text{cr_load} \rightarrow \text{notshared}(\text{ccr}(\text{cre}_1))$,

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

$\mathrm{notshared(cfb(fbe_1)) \land notshared(ccr(cre_2)) \to cfb(fbe_1) \neq ccr(cre_2)}$,
$\mathrm{notshared(cht(hte_1)) \land notshared(cro(roe_2)) \to cht(hte_1) \neq cro(roe_2)}$,
$\mathrm{notshared(cht(hte_1)) \land notshared(cpr(pre_2)) \to cht(hte_1) \neq cpr(pre_2)}$,
$\mathrm{notshared(cht(hte_1)) \land notshared(cdb(dbe_2)) \to cht(hte_1) \neq cdb(dbe_2)}$,
$\mathrm{notshared(cht(hte_1)) \land notshared(ccr(cre_2)) \to cht(hte_1) \neq ccr(cre_2)}$,
$\mathrm{notshared(cro(roe_1)) \land notshared(cpr(pre_2)) \to cro(roe_1) \neq cpr(pre_2)}$,
$\mathrm{notshared(cro(roe_1)) \land notshared(cdb(dbe_2)) \to cro(roe_1) \neq cdb(dbe_2)}$,
$\mathrm{notshared(cro(roe_1)) \land notshared(ccr(cre_2)) \to cro(roe_1) \neq ccr(cre_2)}$,
$\mathrm{notshared(cpr(pre_1)) \land notshared(cdb(dbe_2)) \to cpr(pre_1) \neq cdb(dbe_2)}$,
$\mathrm{notshared(cpr(pre_1)) \land notshared(ccr(cre_2)) \to cpr(pre_1) \neq ccr(cre_2)}$,
$\mathrm{notshared(cpr(pre_1)) \land notshared(cdb(dbe_2)) \to cdb(dbe_1) \neq ccr(cre_2)}$,

$\mathrm{notshared(cfb(fbe_1))}$
$\mathrm{\to trans(cfb(fbe_1), fz_1) =}$
$\mathrm{mkstate(fb_tr(fbe_1, fbc(fz_1)), tac(fz_1), roc(fz_1), prc(fz_1), dbc(fz_1), crc(fz_1))}$,

$\mathrm{notshared(cht(hte_1))}$
$\mathrm{\to trans(cht(hte_1), fz_1) =}$
$\mathrm{mkstate(fbc(fz_1), ta_tr(hte_1, tac(fz_1)), roc(fz_1), prc(fz_1), dbc(fz_1), crc(fz_1))}$,

$\mathrm{notshared(cro(roe_1))}$
$\mathrm{\to trans(cro(roe_1), fz_1) =}$
$\mathrm{mkstate(fbc(fz_1), tac(fz_1), ro_tr(roe_1, roc(fz_1)), prc(fz_1), dbc(fz_1), crc(fz_1))}$,

$\mathrm{notshared(cpr(pre_1))}$
$\mathrm{\to trans(cpr(pre_1), fz_1) =}$
$\mathrm{mkstate(fbc(fz_1), tac(fz_1), roc(fz_1), pr_tr(pre_1, prc(fz_1)), dbc(fz_1), crc(fz_1))}$,

$\mathrm{notshared(cdb(dbe_1))}$
$\mathrm{\to trans(cdb(dbe_1), fz_1) =}$
$\mathrm{mkstate(fbc(fz_1), tac(fz_1), roc(fz_1), prc(fz_1), db_tr(dbe_1, dbc(fz_1)), crc(fz_1))}$,

$\mathrm{notshared(ccr(cre_1))}$
$\mathrm{\to trans(ccr(cre_1), fz_1) =}$
$\mathrm{mkstate(fbc(fz_1), tac(fz_1), roc(fz_1), prc(fz_1), dbc(fz_1), cr_tr(cre_1, crc(fz_1)))}$,

$\mathrm{trans(cfb(fb_sheetfalls), fz_1)}$
$\mathrm{= mkstate(fb_tr(fb_sheetfalls, fbc(fz_1)), ta_tr(ta_load, tac(fz_1)),}$
$\mathrm{roc(fz_1), prc(fz_1), dbc(fz_1), crc(fz_1))}$,

$\mathrm{trans(cht(ta_unload), fz_1)}$
$\mathrm{= mkstate(fbc(fz_1), ta_tr(ta_unload, tac(fz_1)), ro_tr(ro_a1load, roc(fz_1)),}$
$\mathrm{prc(fz_1), dbc(fz_1), crc(fz_1))}$,

$\mathrm{trans(cro(ro_a1unload), fz_1)}$
$\mathrm{= mkstate(fbc(fz_1), tac(fz_1), ro_tr(ro_a1unload, roc(fz_1)),}$
$\mathrm{pr_tr(pr_load, prc(fz_1)), dbc(fz_1), crc(fz_1))}$,

$\mathrm{trans(cpr(pr_unload), fz_1)}$
$\mathrm{= mkstate(fbc(fz_1), tac(fz_1), ro_tr(ro_a2load, roc(fz_1)),}$
$\mathrm{pr_tr(pr_unload, prc(fz_1)), dbc(fz_1), crc(fz_1))}$,

trans(cro(ro_a2unload), fz_1)
= mkstate(fbc(fz_1), tac(fz_1), ro_tr(ro_a2unload, roc(fz_1)),
prc(fz_1), db_tr(db_load, dbc(fz_1)), crc(fz_1)),

trans(cdb(db_unload), fz_1)
= mkstate(fbc(fz_1), tac(fz_1), roc(fz_1),
prc(fz_1), db_tr(db_unload, dbc(fz_1)), cr_tr(cr_load, crc(fz_1))),

trans(ccr(cr_unload), fz_1)
= mkstate(fb_tr(fb_load, fbc(fz_1)), tac(fz_1), roc(fz_1),
prc(fz_1), dbc(fz_1), cr_tr(cr_unload, crc(fz_1))),

expe(cfb(fbe_1), fz_1) $\leftrightarrow$ fb_exp(fbe_1, fbc(fz_1)),
$hte_1 \neq$ ta_load $\rightarrow$ (expe(cht(hte_1), fz_1) $\leftrightarrow$ ta_exp(hte_1, tac(fz_1))),
expe(cro(roe_1), fz_1) $\leftrightarrow$ ro_exp(roe_1, roc(fz_1)),
expe(cpr(pre_1), fz_1) $\leftrightarrow$ pr_exp(pre_1, prc(fz_1)),
expe(cdb(dbe_1), fz_1) $\leftrightarrow$ db_exp(dbe_1, dbc(fz_1)),
expe(ccr(cre_1), fz_1) $\leftrightarrow$ cr_exp(cre_1, crc(fz_1))
end enrich

PROD_CELL_WITH_EVENTLIST =
actualize LIST **with** PROD_CELL **by morphism**
elem $\rightarrow$ event, list $\rightarrow$ elist, nil $\rightarrow$ enil, cons $\rightarrow$ econs, car $\rightarrow$ ecar, cdr $\rightarrow$ ecdr,
append $\rightarrow$ eappend, reverse $\rightarrow$ ereverse, a $\rightarrow$ e, l $\rightarrow$ el, $l_1 \rightarrow el_1$
end actualize

PRODUCTION_CELL =
enrich PROD_CELL_WITH_EVENTLIST **with**
functions trans* : elist $\times$ state $\rightarrow$ state ;
variables el_1: elist;
axioms
trans*(enil, fz_1) = fz_1,
trans*(econs(e, el), fz_1) = trans*(el, trans(e, fz_1))
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_tmvto(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_a1pos(roc(z1)) = ro_amid then
       begin
          el := econs(cro(ro_a1load), el);
          el := econs(cro(ro_a1mv(ro_ain)), el);
          el := econs(cta(ta_tmvto(ta_fb)), el)
       end
  end

else
if e = cro(ro_a1at(ro_amid)) then
  begin
     if ta_tpos(tac(z1)) = ta_ro then
       begin
          el := econs(cro(ro_a1load), el);
          el := econs(cro(ro_a1mv(ro_ain)), el);
          el := econs(cta(ta_tmvto(ta_fb)), el)
       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

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
  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)

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
```

```
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(crc(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

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
```

```
else
if e = ccr(cr_aat(cr_amid)) then
  begin
     if ¬ 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

else
if e = ccr(cr_aat(cr_aup)) then
  if cr_aloaded(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_amvto(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_a1loaded(roc(z1))
     -> (ro_pos(roc(z1)) = ro_a1ta
         -> ro_a1pos(roc(z1)) = ro_atoin))

and (ro_a1loaded(roc(z1))
     -> (ro_pos(roc(z1)) = ro_a1pr
         ->    ro_a1pos(roc(z1)) = ro_ain
            or ro_a1pos(roc(z1)) = ro_atoout))

and (ro_a1loaded(roc(z1))
     -> not ro_pos(roc(z1)) = ro_a1tota)

and (not ro_a1loaded(roc(z1))
     -> (ro_pos(roc(z1)) = ro_a1pr
         -> ro_a1pos(roc(z1)) = ro_atoin))

and (not ro_a1loaded(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_aloaded(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_aloaded(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_aloaded(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_aloaded(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, Forschunszentrum 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. Forschunszentrum 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.