

SOFTWARE VERIFICATION RESEARCH CENTRE
DEPARTMENT OF COMPUTER SCIENCE
THE UNIVERSITY OF QUEENSLAND

Queensland 4072
Australia

TECHNICAL REPORT

No. 94-46

Z Specification of the Production Cell

Anthony MacDonald
David Carrington

October 1994

Phone: +61 7 365 1003

Fax: +61 7 365 1533

Note: Most SVRC technical reports are available via anonymous ftp, from `ftp.cs.uq.edu.au` in the directory `/pub/SVRC/techreports`.

Z Specification of the Production Cell

Anthony MacDonald and David Carrington

Software Verification Research Centre
The University of Queensland
Queensland 4072, Australia

email: {anti, davec}@cs.uq.oz.au

Abstract

This paper uses the Z specification language to specify the Production Cell. The specification models the Production Cell at an abstract level to concisely capture the requirements of the system. The specification uses the structure of the physical system to make the model easy to understand.

1 Introduction

The original production cell can be found in a metal processing plant in Karlsruhe, Germany. Forschungszentrum Informatik (FZI), Karlsruhe, used the production cell as the basis of a study on the use of formal methods for critical software systems[2]. The first step of the study generated a requirements document[3] which attempts to capture, in plain language, the specification of a software controller for the production cell. To simplify the model, the system specified is a simulation of the production cell and the simulator (see figure 1) runs cyclically. The production cell simulation (the system) takes a metal blank as input. The metal blank is transported by a conveyer belt (the feed belt) to an elevating rotary table. The elevating rotary table rotates and rises vertically to present the metal blank for the robot to pick-up. The robot takes the metal blank to a press. The press presses the metal blank and the robot retrieves the metal blank. The robot transports the metal blank to a second conveyer belt (the deposit belt). The deposit belt transports the metal blank to a crane. The crane picks up the metal blank and transports the metal blank to the feed belt and completes the cycle.

The system actions are further complicated by a desire for speed and efficiency. To accommodate these aims, the robot is fitted with two arms. The arms are placed perpendicularly to

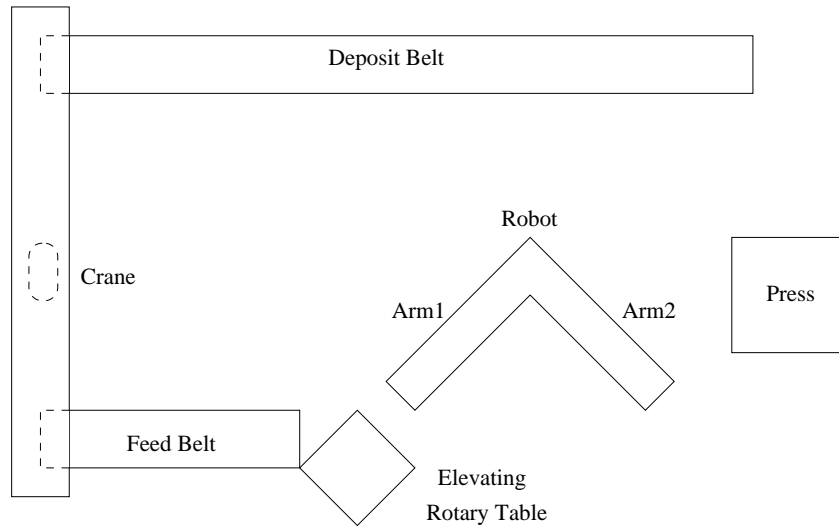


Figure 1: The production cell

each other and must rotate together. The arms can extend/retract and load/unload independently. A second consequence of the aim for speed and efficiency is that the system should be able to handle multiple metal blanks in different parts of the system concurrently.

The Z specification[1, 4] of the system has been built in a bottom-up manner. The first part is the independent specification of each component of the system. The system is separated into the following components:- the feed belt, the elevating rotary table, the robot, the press, the deposit belt and the crane. The independent specification of each component captures the local state information and the allowable operations on the state. The independent specifications do not take into account the relationship between different components. For example, the robot cannot pick up a metal blank from the elevating rotary table if no metal blank is there, however this information is not relevant to the independent specifications. The independent specifications, each with their own local state, are combined to give a total, overall state of the system. The operations from the independent specifications are promoted to apply in the total state and take into consideration the relationships between components. The system contains an overall state and the operations allowed on the state, but the operations are partial and are only guaranteed to succeed when all the pre-conditions are satisfied. The behaviour is not specified when the pre-conditions are not satisfied. The final section of the specification extends the partial operations to total operations. The extension is achieved by considering the possible error cases for each partial operation and specifying the error cases in an error schema for the operation. The final total operation is the joining of the partial operation and the error schema via the schema calculus.

2 The Feed Belt

The feed belt receives a metal blank from the crane¹, transports the metal blank to the other end of the feed belt and delivers the metal blank to the elevating rotary table. The feed belt has been modeled with a high level of abstraction and the following assumptions have been made:-

- The only operations that the feed belt can perform are load or unload operations. The load operation accepts a metal blank onto the feed belt and the unload operation removes a metal blank from the feed belt. All transportation of metal blanks is hidden; if the feed belt is ready to unload then the metal blank has been transported from the loading position to the unloading position.
- The feed belt can contain only one metal blank at a time.

Both assumptions allow for a simpler specification and could be extended with limited work. To enable extra metal blanks on the feed belt, the feed belt could be broken into multiple zones. For example, the first zone could be a loading zone, the second zone a transport zone and the third zone an unloading zone. This simple extension would allow three metal blanks on the feed belt, however if more are required the transport zone could be treated as a sequence of zones. Accompanying the extra positions, an extra operation would be needed to move the metal blanks from zone to zone. The extensions would make the specification both larger and more complex and it can be argued that the conveyer belts are likely to be the fastest components of the system and would not slow the overall system down. Even with the assumption of one metal blank on a conveyer belt at a time, the system can have five metal blanks being processed concurrently.

2.1 The Feed Belt's State

The feed belt's state has one variable and this variable is of type *Component_Loaded* which has two states. The belt is ready to load and is therefore *unloaded* or is ready to unload and hence must have a metal blank on board (*loaded*). The state of the feed belt is initially *unloaded*.

$$\textit{Component_Loaded} ::= \textit{loaded} \mid \textit{unloaded}$$

$\frac{\textit{Feed_Belt}}{\textit{feed_belt_loaded} : \textit{Component_Loaded}}$

$\frac{\textit{Init_Feed_Belt}}{\textit{Feed_Belt}}$ $\textit{feed_belt_loaded} = \textit{unloaded}$

¹The feed belt may also receive a metal blank from outside the system. This is discussed in the section on the cell as a whole (Section 8).

2.2 The Feed Belt's Operations

Building on the simple state of the feed belt, the operations are straight forward and toggle the state. For a *Load_Feed_Belt* operation to occur, the precondition is the feed belt is *unloaded* (ready to load) and the post condition is the feed belt must be *loaded* and hence ready to unload.

$\frac{\text{Load_Feed_Belt}}{\Delta \text{Feed_Belt}}$	$\frac{\text{Unload_Feed_Belt}}{\Delta \text{Feed_Belt}}$
$\text{feed_belt_loaded} = \text{unloaded}$	$\text{feed_belt_loaded} = \text{loaded}$
$\text{feed_belt_loaded}' = \text{loaded}$	$\text{feed_belt_loaded}' = \text{unloaded}$

3 The Elevating Rotary Table

The elevating rotary table transports metal blanks from the feed belt to the robot. To transport metal blanks, the elevating rotary table must move from a position at the end of the feed belt to a position where the robot can remove the metal blanks from the elevating rotary table. Due to the robot's position relative to the feed belt, the table must raise the metal blank vertically and rotate clockwise approximately 1/8 th of a turn. When the metal blank has been removed, the table returns to its initial position at the end of the feed belt.

3.1 The Elevating Rotary Table's State

The state of the elevating rotary table is modeled by *table_position* and *table_loaded*, which record the elevating rotary table's position and whether the elevating rotary table is *loaded* or *unloaded*. The elevating rotary table has been restricted to two positions. The first, *ready_to_load*, is at the end of the feed belt and not rotated. The second, *ready_to_unload*, is at the point where the robot can retrieve the metal blank and is both elevated and rotated. Initially the elevating rotary table is in position to be loaded and is empty.

$$\text{Table_Position} ::= \text{ready_to_load} \mid \text{ready_to_unload}$$

$\frac{\text{ERT}}{\text{table_position} : \text{Table_Position}}$	$\frac{\text{Init_ERT}}{\text{ERT}}$
$\text{table_loaded} : \text{Component_Loaded}$	$\text{table_position} = \text{ready_to_load}$
	$\text{table_loaded} = \text{unloaded}$

3.2 The Elevating Rotary Table's Operations

There are four operations on the elevating rotary table with the first two being related to loading and unloading the elevating rotary table and the second two operations being the move operations. To receive a metal blank, the elevating rotary table must be in position to be loaded and must be *unloaded*. At the conclusion of the operation, *Load_ERT_0*, the elevating rotary table is loaded and its position is unchanged. Similarly, when an *Unload_ERT_0* operation occurs, the elevating rotary table must be in the unload position and be *loaded*, with successful completion implying the elevating rotary table has not moved but is now *unloaded*.

$\overline{Load_ERT_0}$ ΔERT	$\overline{Unload_ERT_0}$ ΔERT
$table_position = ready_to_load$ $table_loaded = unloaded$ $table_position' = table_position$ $table_loaded' = loaded$	$table_position = ready_to_unload$ $table_loaded = loaded$ $table_position' = table_position$ $table_loaded' = unloaded$

The movement of the elevating rotary table has been simplified to two moves, either moving the elevating rotary table to the loading position (next to the feed belt) or to the unloading position (ready for the robot). The simplification of the movement abstracts from the fact that a move includes both left/right rotation and horizontal/vertical translation.

$\overline{Move_ERT_to_Unloading_Position_0}$ ΔERT	$\overline{Move_ERT_to_Loading_Position_0}$ ΔERT
$table_position' = ready_to_unload$ $table_loaded' = table_loaded$	$table_position' = ready_to_load$ $table_loaded' = table_loaded$

4 The Robot

The robot is the most complex part of the production cell simulation and the modeling of the robot's state and operations is the most detailed part of the specification.

4.1 An Arm

The robot has two identical arms. To minimise duplication a generic arm is specified.

$Arm_Extent ::= retracted \mid extended$

An arm has four state variables. The first, *load*, records whether an arm is *loaded* or *unloaded* and an arm is initially *unloaded*. The second, *extent*, records whether the arm is *extended* or

retracted and is initialised to *retracted*. The final two state variables, *load_pos* and *unload_pos* are the set of orientations the arm can be loaded or unloaded at. At instantiation the type of orientation is set as are the values of *load_pos* and *unload_pos*. The arm never changes *load_pos* and *unload_pos* and a delta (Δ) schema is declared that equates these variables for all arm operations.

$\frac{\text{Arm}[\textit{Orientation}]}{\textit{load} : \textit{Component_Loaded}$ $\textit{extent} : \textit{Arm_Extent}$ $\textit{load_pos} : \mathbb{P} \textit{Orientation}$ $\textit{unload_pos} : \mathbb{P} \textit{Orientation}$	$\frac{\Delta \text{Arm}[\textit{Orientation}]}{\text{Arm}[\textit{Orientation}]}$ $\text{Arm}'[\textit{Orientation}]$ <hr style="border: 0.5px solid black;"/> $\textit{load_pos}' = \textit{load_pos}$ $\textit{unload_pos}' = \textit{unload_pos}$
$\frac{\text{Init_Arm}[\textit{Orientation}]}{\text{Arm}[\textit{Orientation}]}$ <hr style="border: 0.5px solid black;"/> $\textit{load} = \textit{unloaded}$ $\textit{extent} = \textit{retracted}$	

Arm extension and retraction always succeed (at this level of abstraction) and change the value of *extent* while leaving *load* unchanged.

$\frac{\text{Extend}[\textit{Orientation}]}{\Delta \text{Arm}[\textit{Orientation}]}$ <hr style="border: 0.5px solid black;"/> $\textit{extent}' = \textit{extended}$ $\textit{load}' = \textit{load}$	$\frac{\text{Retract}[\textit{Orientation}]}{\Delta \text{Arm}[\textit{Orientation}]}$ <hr style="border: 0.5px solid black;"/> $\textit{extent}' = \textit{retracted}$ $\textit{load}' = \textit{load}$
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

An arm can only load and unload at certain orientations, and to perform a *Load* or *Unload* operation the arm must be told the current arm orientation and compare the orientation to the set of allowable load or unload orientations. These operations change only the *load* variable of the arm.

$\frac{\text{Load}[\textit{Orientation}]}{\Delta \text{Arm}[\textit{Orientation}]}$ $\textit{current_pos}? : \textit{Orientation}$ <hr style="border: 0.5px solid black;"/> $\textit{current_pos}? \in \textit{load_pos}$ $\textit{extent} = \textit{extended}$ $\textit{load} = \textit{unloaded}$ $\textit{extent}' = \textit{extent}$ $\textit{load}' = \textit{loaded}$	$\frac{\text{Unload}[\textit{Orientation}]}{\Delta \text{Arm}[\textit{Orientation}]}$ $\textit{current_pos}? : \textit{Orientation}$ <hr style="border: 0.5px solid black;"/> $\textit{current_pos}? \in \textit{unload_pos}$ $\textit{extent} = \textit{extended}$ $\textit{load} = \textit{loaded}$ $\textit{extent}' = \textit{extent}$ $\textit{load}' = \textit{unloaded}$
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

4.2 The Robot's State

The robot has two arms, called *arm1* and *arm2*, and four discrete orientations (see figure 2).

- The first orientation, called *load_arm1*, aligns *arm1* to the elevating rotary table while *arm2* is not aligned with any other production cell component.
- The second orientation, *load_arm2*, aligns *arm2* to the press and *arm1* is not aligned.
- The third orientation, *unload_arm2*, aligns *arm2* for unloading to the deposit belt and *arm1* is not aligned.
- The fourth orientation, *unload_arm1*, aligns *arm1* to the press and *arm2* is not aligned.

$$\textit{Robot_Orientation} ::= \textit{load_arm1} \mid \textit{load_arm2} \mid \textit{unload_arm2} \mid \textit{unload_arm1}$$

<i>Robot</i>
$\textit{robot_orientation} : \textit{Robot_Orientation}$ $\textit{arm1}, \textit{arm2} : \textit{Arm}[\textit{Robot_Orientation}]$
$\textit{arm1.extent} = \textit{extended} \Leftrightarrow \textit{robot_orientation} \in \{\textit{load_arm1}, \textit{unload_arm1}\}$ $\textit{arm1.load_pos} = \{\textit{load_arm1}\}$ $\textit{arm1.unload_pos} = \{\textit{unload_arm1}\}$ $\textit{arm2.extent} = \textit{extended} \Leftrightarrow \textit{robot_orientation} \in \{\textit{load_arm2}, \textit{unload_arm2}\}$ $\textit{arm2.load_pos} = \{\textit{load_arm2}\}$ $\textit{arm2.unload_pos} = \{\textit{unload_arm2}\}$

The initialisation of the robot is done in two stages. The first stage is the specification of the intermediate schema *Init_Robot_0* and the framing schema *Init_Arm_1* and *Init_Arm_2*. The schema *Init_Robot_0* initialises *robot_orientation* the only component of the robot not associated with the arms. The robot's orientation is initially *load_arm1*. The framing schema enables operations from *Arm* to be restricted to a specific arm. The second stage is the the combination of the initialisation operations to give the robot's initialisation. The *Arm* state is hidden as it is not needed at this level, but is present as a result of the promotion of the arm operations.

<i>Init_Robot_0</i>
\textit{Robot} $\textit{robot_orientation} = \textit{load_arm1}$

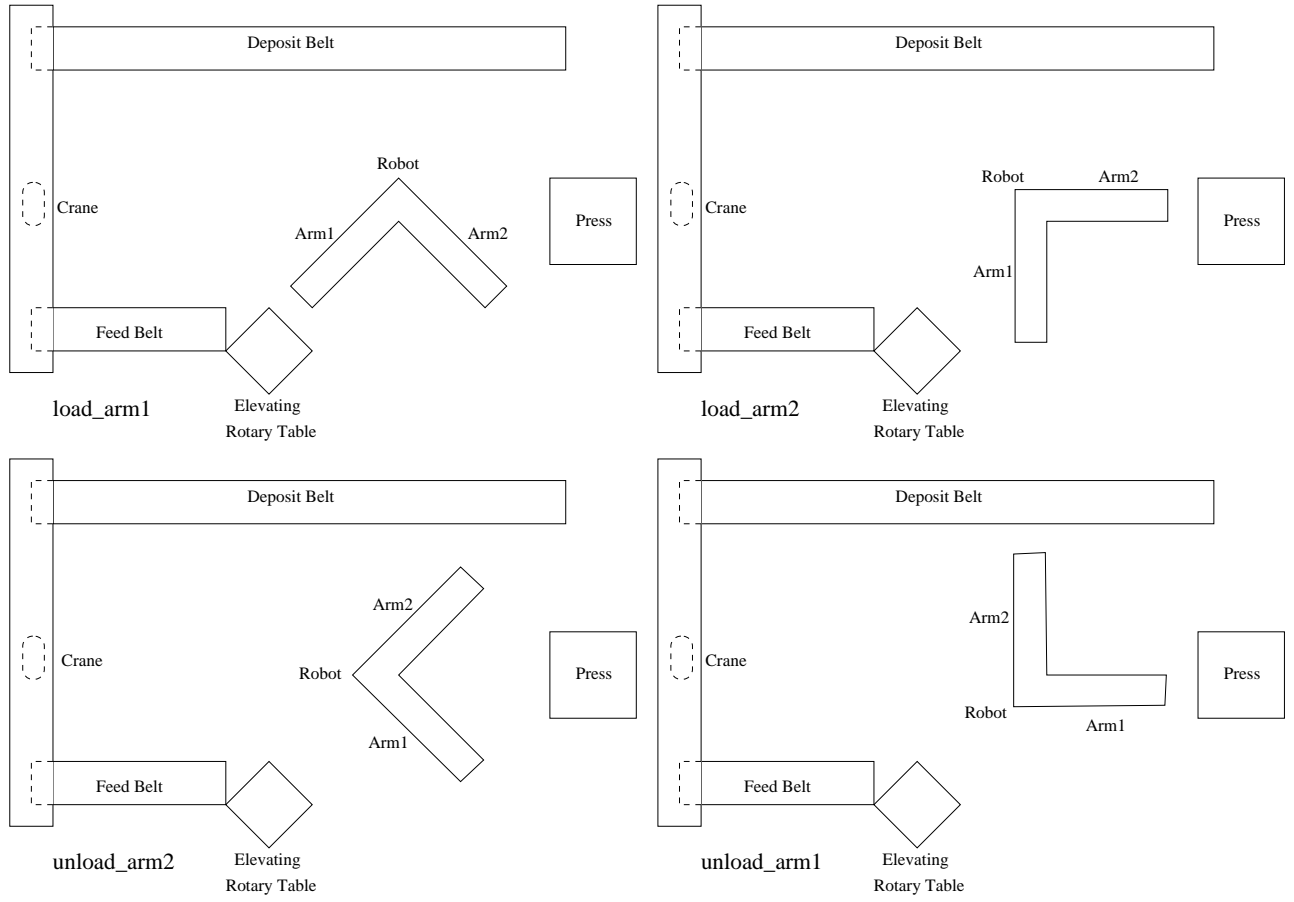
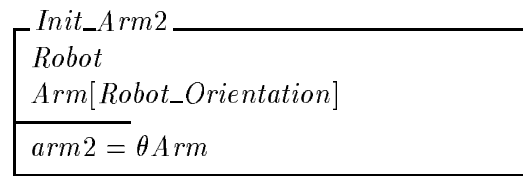
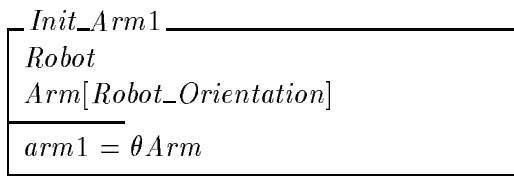


Figure 2: Allowable Robot Orientations



$$\begin{aligned}
 Init_Robot \hat{=} & (Init_Arm1 \wedge Init_Arm[Robot_Orientation]) \setminus (Arm[Robot_Orientation]) \\
 & \wedge (Init_Arm2 \wedge Init_Arm[Robot_Orientation]) \setminus (Arm[Robot_Orientation]) \\
 & \wedge Init_Robot_0
 \end{aligned}$$

4.3 The Robot's Operations

For safety reasons, the robot must not rotate if one or both of the arms is extended. The robot's rotation does not change either the arm loaded state or the arm extent state, but only queries the relevant variables. The robot's new orientation is the value of the input variable *new_pos?*.

$\frac{\text{Rotate_Robot_0}}{\Delta Robot}$ $\text{new_pos?} : Robot_Orientation$
$\text{arm1.extent} = \text{retracted} \wedge \text{arm2.extent} = \text{retracted}$ $\text{arm1'.load} = \text{arm1.load} \wedge \text{arm2'.load} = \text{arm2.load}$ $\text{arm1'.extent} = \text{arm1.extent} \wedge \text{arm2'.extent} = \text{arm2.extent}$ $\text{robot_orientation}' = \text{new_pos?}$

The arm operations need to check the current robot position before loading or unloading can occur. *Generate_Orientation* outputs the current *robot_orientation* which can be piped into the relevant operations and the arm operations do not need to know about the robot's state.

$\frac{\text{Generate_Orientation}}{\Xi Robot}$ $\text{current_pos!} : Robot_Orientation$
$\text{current_pos!} = \text{robot_orientation}$

Arm1_Ops and *Arm2_Ops* are framing schema for the operations on arm1 and arm2 respectively.

$\frac{\text{Arm1_Ops}}{\Delta Robot}$ $\Delta Arm[Robot_Orientation]$	$\frac{\text{Arm2_Ops}}{\Delta Robot}$ $\Delta Arm[Robot_Orientation]$
$\text{arm2}' = \text{arm2}$ $\text{robot_orientation}' = \text{robot_orientation}$ $\text{arm1} = \theta Arm$ $\text{arm1}' = \theta Arm'$	$\text{arm1}' = \text{arm1}$ $\text{robot_orientation}' = \text{robot_orientation}$ $\text{arm2} = \theta Arm$ $\text{arm2}' = \theta Arm'$

The extension and retraction operations for the robot's arms are simply promoted from the *Arm* state with restriction to either arm1 or arm2 and the same hiding as for the initialisation.

The loading and unloading has the addition of the current orientation being piped into the operations from *Generate_Orientation*.

$$\textit{Extend_Arm1_0} \hat{=} (\textit{Arm1_Ops} \wedge \textit{Extend}[\textit{Robot_Orientation}]) \setminus (\Delta \textit{Arm}[\textit{Robot_Orientation}])$$

$$\textit{Extend_Arm2_0} \hat{=} (\textit{Arm2_Ops} \wedge \textit{Extend}[\textit{Robot_Orientation}]) \setminus (\Delta \textit{Arm}[\textit{Robot_Orientation}])$$

$$\textit{Retract_Arm1_0} \hat{=} (\textit{Arm1_Ops} \wedge \textit{Retract}[\textit{Robot_Orientation}]) \setminus (\Delta \textit{Arm}[\textit{Robot_Orientation}])$$

$$\textit{Retract_Arm2_0} \hat{=} (\textit{Arm2_Ops} \wedge \textit{Retract}[\textit{Robot_Orientation}]) \setminus (\Delta \textit{Arm}[\textit{Robot_Orientation}])$$

$$\textit{Load_Arm1_0} \hat{=} \textit{Generate_Orientation} \gg$$

$$((\textit{Arm1_Ops} \wedge \textit{Load}[\textit{Robot_Orientation}]) \setminus (\Delta \textit{Arm}[\textit{Robot_Orientation}])))$$

$$\textit{Load_Arm2_0} \hat{=} \textit{Generate_Orientation} \gg$$

$$((\textit{Arm2_Ops} \wedge \textit{Load}[\textit{Robot_Orientation}]) \setminus (\Delta \textit{Arm}[\textit{Robot_Orientation}])))$$

$$\textit{Unload_Arm1_0} \hat{=} \textit{Generate_Orientation} \gg$$

$$((\textit{Arm1_Ops} \wedge \textit{Unload}[\textit{Robot_Orientation}]) \setminus (\Delta \textit{Arm}[\textit{Robot_Orientation}])))$$

$$\textit{Unload_Arm2_0} \hat{=} \textit{Generate_Orientation} \gg$$

$$((\textit{Arm2_Ops} \wedge \textit{Unload}[\textit{Robot_Orientation}]) \setminus (\Delta \textit{Arm}[\textit{Robot_Orientation}])))$$

5 The Press

The press receives metal blanks from robot arm1, closes to press the metal blank and opens for arm2 to remove the metal blank.

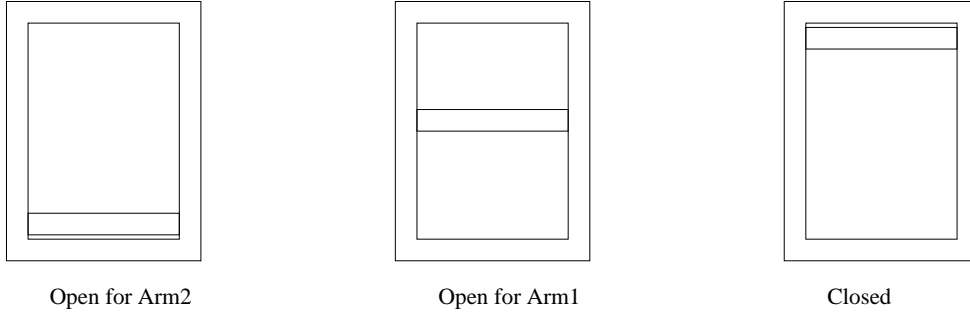


Figure 3: The three press positions

5.1 The Press's State

For technical reasons the robot arms are placed at different heights. A side-effect of the robot arms being at different heights is that the press must have three positions (see figure 3). The

first position of the press is *open_for_arm2* at which point the press is open to the lower position. The second position is *open_for_arm1* which has the press at the middle position. The final position *closed*, is at the upper position. The press also can either be *loaded* or *unloaded*.

$$\text{Press_Position} ::= \text{closed} \mid \text{open_for_arm1} \mid \text{open_for_arm2}$$

Initially the press must be empty and the press position does not matter.

$\frac{}{\text{press_position} : \text{Press_Position}}$ $\text{press_loaded} : \text{Component_Loaded}$	$\frac{}{\text{Init_Press}}$ $\text{press_loaded} = \text{unloaded}$
--------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------

5.2 The Press's Operations

The press has two operations for transferring metal blanks to and from the press. The first, *Load_Press_0*, is for the press to interact with arm1. The press must be *open_for_arm1* and the press must be empty. The result of *Load_Press_0* is for a metal blank to be inside the press with the press position unchanged. The second, *Unload_Press_0*, is for the press to interact with arm2. The press must be in position *open_for_arm2* and the press must have a metal blank inside. The press is empty after the operation and the press position has not changed.

$\frac{}{\Delta \text{Press}}$ $\text{press_position} = \text{open_for_arm1}$ $\text{press_loaded} = \text{unloaded}$ $\text{press_position}' = \text{press_position}$ $\text{press_loaded}' = \text{loaded}$	$\frac{}{\Delta \text{Press}}$ $\text{press_position} = \text{open_for_arm2}$ $\text{press_loaded} = \text{loaded}$ $\text{press_position}' = \text{press_position}$ $\text{press_loaded}' = \text{unloaded}$
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

There is one operation to move the press, *Move_Press_0*, which sets the press position to the supplied position. The operation does not affect the *press_loaded* state.

$\frac{}{\Delta \text{Press}}$ $\text{new_pos?} : \text{Press_Position}$ $\text{press_position}' = \text{new_pos?}$ $\text{press_loaded}' = \text{press_loaded}$

6 The Deposit Belt

The deposit belt receives a metal blank from the robot (at the loading position) and transports the metal blank the length of the deposit belt to the unloading position. At the unloading position, the metal blank is retrieved by a crane. The same assumptions and abstractions made for the feed belt are assumed here. The assumptions without explanation are that the deposit belt can only contain one metal blank at any one time and the only operations allowed on the deposit belt are to receive a metal blank and to deliver a metal blank. The transporting of the metal blank by the deposit belt is implicit.

6.1 The Deposit Belt's State

The deposit belt's state is an instance of the type *Component_Loaded* and allows two possible states for the deposit belt. The first, *unloaded*, means the deposit belt is in position to receive a metal blank from the robot and is not *loaded*. The second position, *loaded*, means the deposit belt is in position to unload to the crane and the deposit belt is *loaded*. Initially the deposit belt is *unloaded*.

$\frac{\textit{Deposit_Belt}}{\textit{deposit_belt_loaded} : \textit{Component_Loaded}}$

$\frac{\textit{Init_Deposit_Belt}}{\textit{Deposit_Belt}}$
$\textit{deposit_belt_loaded} = \textit{unloaded}$

6.2 The Deposit Belt's Operations

As for the feed belt, two operations manipulate the deposit belt. *Load_Deposit_Belt_0* expects the deposit belt to be empty and ready to receive a metal blank and leaves the deposit belt *loaded* and ready to deliver a metal blank to the crane. *Unload_Deposit_Belt_0* is the complementary operation which unloads the *loaded* deposit belt leaving the deposit belt ready to be *loaded* again.

$\frac{\textit{Load_Deposit_Belt_0}}{\Delta \textit{Deposit_Belt}}$
$\textit{deposit_belt_loaded} = \textit{unloaded}$ $\textit{deposit_belt_loaded}' = \textit{loaded}$

$\frac{\textit{Unload_Deposit_Belt_0}}{\Delta \textit{Deposit_Belt}}$
$\textit{deposit_belt_loaded} = \textit{loaded}$ $\textit{deposit_belt_loaded}' = \textit{unloaded}$

7 The Crane

The crane picks up metal blanks at the deposit belt, transports the metal blank to the feed belt and deposits the metal blank onto the feed belt. The crane can return to the deposit belt and the cycle can be repeated. In specifying the crane, the relative heights of the feed and deposit belts² have been ignored. It is assumed a load operation on the deposit belt deals internally with the raising and lowering of the crane to get the blank. A similar assumption is made for the unload operation onto the feed belt. Assuming the raising and lowering of the crane is internal may seem inconsistent with the explicit specification of the robot, however the robot is the most complex part of the production cell and has the greatest interaction with the rest of the system. Conversely, the crane is relatively simple and has simple interactions with the system.

7.1 The Crane's State

The crane's state has two variables. The first, *crane_position*, records whether the crane is *over_deposit_belt* or *over_feed_belt*. The second, *crane_loaded*, records whether the crane is *loaded* or the crane is *unloaded*. Initially the crane is *unloaded*.

$$\text{Crane_Position} ::= \text{over_deposit_belt} \mid \text{over_feed_belt}$$

$\frac{\text{Crane}}{\text{crane_position} : \text{Crane_Position} \\ \text{crane_loaded} : \text{Component_Loaded}}$	$\frac{\text{Init_Crane}}{\text{Crane} \\ \text{crane_loaded} = \text{unloaded}}$
---------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------

7.2 The Crane's Operations

The crane has three possible operations. The first, *Load_Crane_0*, picks up a metal blank from the deposit belt. To pick up a metal blank, the crane must be both *over_deposit_belt* and *unloaded*. When the crane has picked up the metal blank, the crane's position is unchanged but the crane is now *loaded*. The second operation, *Unload_Crane_0*, is similar. The crane deposits a metal blank onto the feed belt. The pre-condition of the unload operation is the crane must be in position, *over_feed_belt*, and the crane must be *loaded*. The crane is not

²The belts are at differing heights for technical reasons relating to the robot's arms.

moved after the unload operation but the crane is now *unloaded*.

$\frac{}{\Delta Crane}$ $crane_position = over_deposit_belt$ $crane_loaded = unloaded$ $crane_position' = crane_position$ $crane_loaded' = loaded$	$\frac{}{\Delta Crane}$ $crane_position = over_feed_belt$ $crane_loaded = loaded$ $crane_position' = crane_position$ $crane_loaded' = unloaded$
-----------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------

The third operation deals with moving the crane between the two possible positions and does not change the crane load state, *crane_loaded*.

$\frac{}{\Delta Crane}$ $new_pos? : Crane_Position$ $crane_position' = new_pos?$ $crane_loaded' = crane_loaded$

8 The Production Cell

The specification of the system needs an overall state so the operations which interact can be specified. This process is called promotion. The first step is the promotion of the individual states to be part of the total state, *Cell*. The operations specified are partial operations on the state. The operations are not total because the behaviour of each operation is unspecified when its pre-condition is not satisfied.

$$Cell \hat{=} Feed_Belt \wedge ERT \wedge Robot \wedge Press \wedge Deposit_Belt \wedge Crane$$

The initialisation of the *Cell* is a conjunction of each component's initialisation.

$$Init_Cell \hat{=} Init_Feed_Belt \wedge Init_ERT \wedge Init_Robot \wedge Init_Press \\ \wedge Init_Deposit_Belt \wedge Init_Crane$$

To simplify the promotion process, sub-states of *Cell* are declared. The sub-states are equated within the partial operations. Equating a sub-state within an operation is equivalent to equating each of the state variables within the sub-state in the operation. The use of sub-states is purely for aesthetics. In the case below, only the state variables belonging to the feed

belt and the elevating rotary table may be changed³.

$$Cell_0 \cong Cell \setminus (Feed_Belt, ERT)$$

The partial operation to load the elevating rotary table, $Load_ERT_1$, is a combination of the $Unload_Feed_Belt$ and the $Load_ERT_0$ operations. The operation moves a metal blank from the feed belt to the elevating rotary table and hence changes the load states of both the feed belt and the elevating rotary table.

$Load_ERT_1$
$\Delta Cell$
$Unload_Feed_Belt$
$Load_ERT_0$
<hr/> $\theta Cell_0' = \theta Cell_0$

The next two operations move the elevating rotary table and do not change any state variables except those related to the elevating rotary table.

$$Cell_1 \cong Cell \setminus (ERT)$$

The first operation, $Move_ERT_to_Loading_Position_1$, simply promotes the elevating rotary table's operation, $Move_ERT_to_Loading_Position_0$, to apply over the whole state. The second operation, $Move_ERT_to_Unloading_Position_1$, while not interacting with another operation, needs an extra precondition to be satisfied before the operation can proceed. The operation requires arm1 of the robot to be retracted if the robot is in position $load_arm1$ or if arm1 is extended at $load_arm1$, the arm must be unloaded. The pre-condition ensures there is no collision between the loaded robot and the elevating rotary table.

$Move_ERT_to_Loading_Position_1$
$\Delta Cell$
$Move_ERT_to_Loading_Position_0$
<hr/> $\theta Cell_1' = \theta Cell_1$

³The hiding for $Cell_0$ should actually contain the variables being hidden in brackets rather than the state schema to which the variables belong. However, this is visually messy and the hiding of the state is equivalent.

$\text{Move_ERT_to_Unloading_Position_1}$
ΔCell
$\text{Move_ERT_to_Unloading_Position_0}$
$\text{robot_orientation} = \text{load_arm1} \Rightarrow$ $(\text{arm1.extent} = \text{retracted} \vee$ $(\text{arm1.extent} = \text{extended} \wedge \text{arm1.load} = \text{unloaded}))$
$\theta \text{Cell_1}' = \theta \text{Cell_1}$

The next five operations change only the robot's state.

$$\text{Cell_2} \cong \text{Cell} \setminus (\text{Robot})$$

The operations to extend the robot's arms, *Extend_Arm1_1* and *Extend_Arm2_1*, promote the operations from the individual state and add preconditions that ensure an arm does not collide with the press or a loaded arm does not collide with another metal blank.

Extend_Arm1_1
ΔCell
Extend_Arm1_0
$(\text{robot_orientation} = \text{load_arm1} \wedge \text{table_position} = \text{ready_to_unload} \wedge$ $\text{table_loaded} = \text{loaded} \Rightarrow \text{arm1.load} = \text{unloaded})$
$(\text{robot_orientation} = \text{unload_arm1} \wedge \text{press_position} = \text{open_for_arm1} \wedge$ $\text{arm1.load} = \text{loaded} \Rightarrow \text{press_loaded} = \text{unloaded})$
$\theta \text{Cell_2}' = \theta \text{Cell_2}$

Extend_Arm2_1
ΔCell
Extend_Arm2_0
$(\text{robot_orientation} = \text{load_arm2} \wedge \text{press_position} = \text{open_for_arm2} \wedge$ $\text{arm2.load} = \text{loaded} \Rightarrow \text{press_loaded} = \text{unloaded})$
$(\text{robot_orientation} = \text{unload_arm2} \wedge \text{arm2.load} = \text{loaded} \Rightarrow$ $\text{deposit_belt_loaded} = \text{unloaded})$
$\theta \text{Cell_2}' = \theta \text{Cell_2}$

The operations to retract the robot's arms and to rotate the robot are simpler and just involve promotion of the relevant operation to the total state.

$$\begin{array}{|l} \hline \textit{Rotate_Robot_1} \\ \hline \Delta \textit{Cell} \\ \textit{Rotate_Robot_0} \\ \hline \theta \textit{Cell_2}' = \theta \textit{Cell_2} \\ \hline \end{array}
 \quad
 \begin{array}{|l} \hline \textit{Retract_Arm1_1} \\ \hline \Delta \textit{Cell} \\ \textit{Retract_Arm1_0} \\ \hline \theta \textit{Cell_2}' = \theta \textit{Cell_2} \\ \hline \end{array}
 \quad
 \begin{array}{|l} \hline \textit{Retract_Arm2_1} \\ \hline \Delta \textit{Cell} \\ \textit{Retract_Arm2_0} \\ \hline \theta \textit{Cell_2}' = \theta \textit{Cell_2} \\ \hline \end{array}$$

To load arm1, state variables in both the robot and the elevating rotary table must be changed as the load state of each is changed with the transfer of the metal blank.

$$\textit{Cell_3} \hat{=} \textit{Cell} \setminus (\textit{Robot}, \textit{ERT})$$

The operation, $\textit{Load_Arm1_1}$ is a combination of $\textit{Load_Arm1_0}$ and $\textit{Unload_ERT_0}$.

$$\begin{array}{|l} \hline \textit{Load_Arm1_1} \\ \hline \Delta \textit{Cell} \\ \textit{Load_Arm1_0} \\ \textit{Unload_ERT_0} \\ \hline \theta \textit{Cell_3}' = \theta \textit{Cell_3} \\ \hline \end{array}$$

Unloading arm1 and loading arm2 involve both the robot state and the press state.

$$\textit{Cell_4} \hat{=} \textit{Cell} \setminus (\textit{Robot}, \textit{Press})$$

To unload arm1, the operation $\textit{Unload_Arm1_1}$ is a combination of $\textit{Unload_Arm1_0}$ and $\textit{Load_Press_0}$. The loading of arm2, $\textit{Load_Arm2_1}$, is $\textit{Load_Arm2_0}$ and $\textit{Unload_Press_0}$ combined.

$$\begin{array}{|l} \hline \textit{Unload_Arm1_1} \\ \hline \Delta \textit{Cell} \\ \textit{Unload_Arm1_0} \\ \textit{Load_Press_0} \\ \hline \theta \textit{Cell_4}' = \theta \textit{Cell_4} \\ \hline \end{array}
 \quad
 \begin{array}{|l} \hline \textit{Load_Arm2_1} \\ \hline \Delta \textit{Cell} \\ \textit{Load_Arm2_0} \\ \textit{Unload_Press_0} \\ \hline \theta \textit{Cell_4}' = \theta \textit{Cell_4} \\ \hline \end{array}$$

Similarly, the unloading of arm2 depends on both the robot and deposit belt states.

$$\textit{Cell_5} \hat{=} \textit{Cell} \setminus (\textit{Robot}, \textit{Deposit_Belt})$$

The operation $\textit{Unload_Arm2_1}$ includes both $\textit{Unload_Arm2_0}$ from the robot's operations and $\textit{Load_Deposit_Belt_0}$ from the deposit belt's operations.

<i>Unload_Arm2_1</i>
$\Delta Cell$
<i>Unload_Arm2_0</i>
<i>Load_Deposit_Belt_0</i>
$\theta Cell_{5'} = \theta Cell_5$

The next operation is for movement of the press and only changes the press state.

$$Cell_6 \cong Cell \setminus (Press)$$

The operation *Move_Press_1* involves promotion of the relevant operation from the press state. An added pre-condition on the operation is that the robot's arms, if at the press (*unload_arm1* for arm1 and *load_arm2* for arm2), must be retracted before the press can move. The extra pre-condition is to avoid the press shutting on or crushing a robot arm.

<i>Move_Press_1</i>
$\Delta Cell$
<i>Move_Press_0</i>
$robot_orientation = unload_arm1 \Rightarrow arm1.extent = retracted$
$robot_orientation = load_arm2 \Rightarrow arm2.extent = retracted$
$\theta Cell_{6'} = \theta Cell_6$

The crane picks up metal blanks from the deposit belt. To do this, the crane must perform a *Load_Crane_0* operation and the deposit belt must perform a *Unload_Deposit_Belt_0* operation. The resulting operation over the *Cell* state is called *Load_Crane_1*.

$$Cell_7 \cong Cell \setminus (Deposit_Belt, Crane)$$

<i>Load_Crane_1</i>
$\Delta Cell$
<i>Unload_Deposit_Belt_0</i>
<i>Load_Crane_0</i>
$\theta Cell_{7'} = \theta Cell_7$

The operation to move the crane changes only the crane state. The operation *Move_Crane_1* is produced by promoting of the equivalent operation on the crane state.

$$Cell_8 \cong Cell \setminus (Crane)$$

<i>Move_Crane_1</i>
$\Delta Cell$
<i>Move_Crane_0</i>
$\theta Cell_{S'} = \theta Cell_S$

Unloading the crane requires a relationship between the crane and the feed belt. The crane performs an *Unload_Crane_0* operation and the feed belt performs a *Load_Feed_Belt* operation. The operation *Unload_Crane_1*, the combination of the two operations, changes elements of the crane state and the feed belt state.

$$Cell_9 \cong Cell \setminus (Feed_Belt, Crane)$$

<i>Unload_Crane_1</i>
$\Delta Cell$
<i>Unload_Crane_0</i>
<i>Load_Feed_Belt</i>
$\theta Cell_{S'} = \theta Cell_9$

The *Cell* as a system is closed at this point, and an operation to add metal blanks is needed. The simplest place to do this is onto the feed belt. *Add_Blank_to_Cell* is achieved by performing a *Load_Feed_Belt* operation and assuming an outside source provides the metal blank.

$$Cell_{10} \cong Cell \setminus (Feed_Belt)$$

<i>Add_Blank_to_Cell</i>
$\Delta Cell$
<i>Load_Feed_Belt</i>
$\theta Cell_{10'} = \theta Cell_{10}$

9 The Error Operations

The operations on the *Cell* defined in the previous section using promotion are only partial operations. To make the partial operations total, the error cases for each partial operation must be considered.

Report is a message output by an operation when it terminates. If the operation was successful the message is *ok*, however if the operation fails, i.e. satisfied one of the error conditions, the message explains the failure.

$$\begin{aligned} \textit{Report} ::= & \textit{ok} \mid \textit{component_already_loaded} \mid \textit{component_already_unloaded} \\ & \mid \textit{wrong_ert_position} \mid \textit{wrong_robot_orientation} \mid \textit{wrong_press_position} \\ & \mid \textit{wrong_crane_position} \mid \textit{feed_belt_not_ready} \mid \textit{deposit_belt_not_ready} \\ & \mid \textit{arm_extended} \mid \textit{arm_retracted} \mid \textit{avoid_collision_between_blanks} \\ & \mid \textit{avoid_collision_arm_press} \end{aligned}$$

The total operations in the final section take the form:-

$$\textit{Total operation} \hat{=} (\textit{partial operation} \wedge \textit{Success}) \vee \textit{error operation}$$

The *Success* schema below enables the total operations to have a specified output. The *Error* schema is included in the error schemas for each partial operation and is typographically simplifying.

$$\textit{Success} \hat{=} [r! : \textit{Report} \mid r! = \textit{ok}] \quad \textit{Error} \hat{=} [\exists \textit{Cell}; r! : \textit{Report}]$$

Each error schema matches one of the partial operations. The name of the error schema is the name of the partial schema with the number extension replaced with the string '*Error*'. For each total operation, there may be more than one possible error. The error schema contains each possible case in the form:-

$$\textit{error condition(s)} \Rightarrow r! = \textit{appropriate error message}$$

Loading the elevating rotary table can fail for several reasons. The first reason is the elevating rotary table is already loaded, the second reason is the elevating rotary table is in the wrong position and the third reason is the feed belt is not able to deliver a metal blank.

<i>Load_ERT_Error</i>
<i>Error</i>
<i>table_loaded = loaded</i> \Rightarrow <i>r!</i> = <i>component_already_loaded</i>
<i>table_position = ready_to_unload</i> \Rightarrow <i>r!</i> = <i>wrong_ert_position</i>
<i>feed_belt_loaded = unloaded</i> \Rightarrow <i>r!</i> = <i>feed_belt_not_ready</i>

The elevating rotary table cannot move to the unloading position if the robot's arm, in this case arm1, is extended towards the elevating rotary table and the arm is *loaded*. This is to avoid a collision between the elevating rotary table and the robot's arm.

<i>Move_ERT_to_Unloading_Position_Error</i>
<i>Error</i>
$arm1.extent = extended \wedge robot_orientation = load_arm1 \wedge$ $arm1.load = loaded \Rightarrow r! = avoid_collision_between_blanks$

The only restriction on robot rotation is the extension/retraction state of each of the robot's arms. The robot rotate operation fails if either of the robot's arms is extended.

<i>Rotate_Robot_Error</i>
<i>Error</i>
$arm1.extent = extended \vee arm2.extent = extended \Rightarrow r! = arm_extended$

Arm1 cannot be extended if the robot is at orientation, *load_arm2* or *unload_arm2*. At orientation, *load_arm1*, arm1 cannot be extended if the elevating rotary table is in position to be unloaded and both the elevating rotary table and arm1 is loaded. This condition avoids collisions between the metal blank on the elevating rotary table and the metal blank held by arm1. At *unload_arm1*, the arm cannot extend if the press is not in the correct position to receive arm1 or if the press is in the correct position and the press and arm1 are loaded. As with the previous case, this error case avoids collisions between arm1 and the press or a loaded arm1 and a loaded press.

<i>Extend_Arm1_Error</i>
<i>Error</i>
$robot_orientation \notin \{load_arm1, unload_arm1\} \Rightarrow r! = wrong_robot_orientation$ $robot_orientation = load_arm1 \Rightarrow$ $(table_position = ready_to_unload \wedge table_loaded = loaded \wedge$ $arm1.load = loaded \Rightarrow r! = avoid_collision_between_blanks)$ $robot_orientation = unload_arm1 \Rightarrow$ $(press_position \neq open_for_arm1 \Rightarrow r! = wrong_press_position \wedge$ $press_position = open_for_arm1 \Rightarrow$ $(press_loaded = loaded \wedge arm1.load = loaded \Rightarrow$ $r! = avoid_collision_between_blanks))$

Arm1 cannot be loaded if it is not at *load_arm1* or if it is retracted. Arm1 cannot be loaded when the arm is already loaded. If the arm is in a correct state, the arm cannot be loaded if the elevating rotary table is in the wrong position or the elevating rotary table is unloaded (but in the correct position).

<i>Load_Arm1_Error</i>
<i>Error</i>
$ \begin{aligned} &robot_orientation \neq load_arm1 \Rightarrow r! = wrong_robot_orientation \\ &robot_orientation = load_arm1 \Rightarrow \\ &\quad (arm1.extent = retracted \Rightarrow r! = arm_retracted \wedge \\ &\quad\quad arm1.extent = extended \Rightarrow \\ &\quad\quad\quad (arm1.load = loaded \Rightarrow r! = component_already_loaded \wedge \\ &\quad\quad\quad\quad arm1.load = unloaded \Rightarrow \\ &\quad\quad\quad\quad\quad (table_position = ready_to_load \Rightarrow r! = wrong_ert_position \wedge \\ &\quad\quad\quad\quad\quad\quad table_position = ready_to_unload \Rightarrow \\ &\quad\quad\quad\quad\quad\quad\quad (table_loaded = unloaded \Rightarrow \\ &\quad\quad\quad\quad\quad\quad\quad\quad r! = component_already_unloaded)))))) \end{aligned} $

The unloading of arm1 fails if the arm is not at *unload_arm1*. If the robot is oriented at *unload_arm1*, the arm must be extended and the press must be open for arm1. The unload operation can still fail at this point if the arm is already unloaded or if the press is already loaded.

<i>Unload_Arm1_Error</i>
<i>Error</i>
$ \begin{aligned} &robot_orientation \neq unload_arm1 \Rightarrow r! = wrong_robot_orientation \\ &robot_orientation = unload_arm1 \Rightarrow \\ &\quad (arm1.extent = retracted \Rightarrow r! = arm_retracted \wedge \\ &\quad\quad arm1.extent = extended \Rightarrow \\ &\quad\quad\quad (press_position \neq open_for_arm1 \Rightarrow r! = wrong_press_position \wedge \\ &\quad\quad\quad\quad press_position = open_for_arm1 \Rightarrow \\ &\quad\quad\quad\quad\quad (arm1.load = unloaded \Rightarrow r! = component_already_unloaded \wedge \\ &\quad\quad\quad\quad\quad\quad arm1.load = loaded \Rightarrow \\ &\quad\quad\quad\quad\quad\quad\quad (press_loaded = loaded \Rightarrow \\ &\quad\quad\quad\quad\quad\quad\quad\quad r! = avoid_collision_between_blanks)))))) \end{aligned} $

Like arm1, arm2 cannot be extended if the robot is not in a suitable orientation. At orientation *load_arm2*, the press must be open for arm2 and a collision must be avoided between a loaded arm2 and a loaded press. Similarly at orientation *unload_arm2*, a loaded arm2 must not collide with a loaded deposit belt.

Extend_Arm2_Error

Error

$robot_orientation \notin \{load_arm2, unload_arm2\} \Rightarrow r! = wrong_robot_orientation$
 $robot_orientation = load_arm2 \Rightarrow$
 $(press_position \neq open_for_arm2 \Rightarrow r! = wrong_press_position \wedge$
 $press_position = open_for_arm2 \Rightarrow$
 $(press_loaded = loaded \wedge arm2.load = loaded \Rightarrow$
 $r! = avoid_collision_between_blanks))$
 $robot_orientation = unload_arm2 \Rightarrow$
 $(arm2.load = loaded \wedge deposit_belt_loaded = loaded \Rightarrow$
 $r! = avoid_collision_between_blanks)$

The loading of arm2 fails if the robot is in the wrong orientation. In the correct orientation, the loading still fails if the arm is retracted, the press is in the wrong position, the arm is already loaded or the press is empty.

Load_Arm2_Error

Error

$robot_orientation \neq load_arm2 \Rightarrow r! = wrong_robot_orientation$
 $robot_orientation = load_arm2 \Rightarrow$
 $(arm2.extent = retracted \Rightarrow r! = arm_retracted \wedge$
 $arm2.extent = extended \Rightarrow$
 $(press_position \neq open_for_arm2 \Rightarrow r! = wrong_press_position \wedge$
 $press_position = open_for_arm2 \Rightarrow$
 $(arm2.load = loaded \Rightarrow r! = component_already_loaded \wedge$
 $arm2.load = unloaded \Rightarrow$
 $(press_loaded = unloaded \Rightarrow$
 $r! = component_already_unloaded))))$

As in loading of arm2, the unloading of arm2 fails when the robot is incorrectly oriented and the arm is retracted. The operation also fails if the arm is already unloaded or if both the arm and the deposit belt are loaded. The last condition avoids a collision between metal blanks.

Unload_Arm2_Error

Error

$robot_orientation \neq unload_arm2 \Rightarrow r! = wrong_robot_orientation$
 $robot_orientation = unload_arm2 \Rightarrow$
 $(arm2.extent = retracted \Rightarrow r! = arm_retracted \wedge$
 $arm2.extent = extended \Rightarrow$
 $(arm2.load = unloaded \Rightarrow r! = component_already_unloaded \wedge$
 $arm2.load = loaded \Rightarrow$
 $(deposit_belt_loaded = loaded \Rightarrow$
 $r! = avoid_collision_between_blanks)))$

The press can only be moved if neither of the robot's arms are extended into the press.

Move_Press_Error

Error

$robot_orientation = unload_arm1 \wedge arm1.extent = extended \Rightarrow$
 $r! = avoid_collision_arm_press$
 $robot_orientation = load_arm2 \wedge arm2.extent = extended \Rightarrow$
 $r! = avoid_collision_arm_press$

The crane cannot pick up a metal blank when the crane is over the feed belt, the crane is already loaded or the deposit belt has not delivered a metal blank.

Load_Crane_Error

Error

$crane_position = over_feed_belt \Rightarrow r! = wrong_crane_position$
 $crane_position = over_deposit_belt \Rightarrow$
 $(crane_loaded = loaded \Rightarrow r! = component_already_loaded \wedge$
 $crane_loaded = unloaded \Rightarrow$
 $(deposit_belt_loaded = unloaded \Rightarrow r! = deposit_belt_not_ready))$

The crane cannot deposit a metal blank if the crane is not over the feed belt, if the crane is already unloaded or the feed belt is not ready to receive a new metal blank.

<i>Unload_Crane_Error</i>
<i>Error</i>
$ \begin{aligned} & crane_position = over_deposit_belt \Rightarrow r! = wrong_crane_position \\ & crane_position = over_feed_belt \Rightarrow \\ & \quad (crane_loaded = unloaded \Rightarrow r! = component_already_unloaded \wedge \\ & \quad crane_loaded = loaded \Rightarrow \\ & \quad \quad (feed_belt_loaded = loaded \Rightarrow r! = feed_belt_not_ready)) \end{aligned} $

The only possible error in adding a metal blank to the system is the feed belt not being ready to receive a new metal blank.

<i>Add_Blank_Error</i>
<i>Error</i>
$feed_belt_loaded = loaded \Rightarrow r! = feed_belt_not_ready$

10 The Total Operations

The total operations are robust versions of the partial operations. They combine the partial operations conjoined to the *Success* schema with the corresponding error schema. Each operation outputs a message, either *ok* or an indication of an error condition.

$$\begin{aligned}
Load_ERT &\hat{=} (Load_ERT_1 \wedge Success) \vee Load_ERT_Error \\
Move_ERT_to_Unloading_Position &\hat{=} (Move_ERT_to_Unloading_Position_1 \wedge Success) \\
&\quad \vee Move_ERT_to_Unloading_Position_Error \\
Move_ERT_to_Loading_Position &\hat{=} (Move_ERT_to_Loading_Position_1 \wedge Success) \\
Rotate_Robot &\hat{=} (Rotate_Robot_1 \wedge Success) \vee Rotate_Robot_Error \\
Extend_Arm1 &\hat{=} (Extend_Arm1_1 \wedge Success) \vee Extend_Arm1_Error \\
Retract_Arm1 &\hat{=} (Retract_Arm1_1 \wedge Success) \\
Load_Arm1 &\hat{=} (Load_Arm1_1 \wedge Success) \vee Load_Arm1_Error \\
Unload_Arm1 &\hat{=} (Unload_Arm1_1 \wedge Success) \vee Unload_Arm1_Error \\
Extend_Arm2 &\hat{=} (Extend_Arm2_1 \wedge Success) \vee Extend_Arm2_Error \\
Retract_Arm2 &\hat{=} (Retract_Arm2_1 \wedge Success) \\
Load_Arm2 &\hat{=} (Load_Arm2_1 \wedge Success) \vee Load_Arm2_Error \\
Unload_Arm2 &\hat{=} (Unload_Arm2_1 \wedge Success) \vee Unload_Arm2_Error \\
Move_Press &\hat{=} (Move_Press_1 \wedge Success) \vee Move_Press_Error
\end{aligned}$$

$$\text{Load_Crane} \hat{=} (\text{Load_Crane_1} \wedge \text{Success}) \vee \text{Load_Crane_Error}$$
$$\text{Move_Crane} \hat{=} (\text{Move_Crane_1} \wedge \text{Success})$$
$$\text{Unload_Crane} \hat{=} (\text{Unload_Crane_1} \wedge \text{Success}) \vee \text{Unload_Crane_Error}$$
$$\text{Add_Blank} \hat{=} (\text{Add_Blank_to_Cell} \wedge \text{Success}) \vee \text{Add_Blank_Error}$$

11 Conclusion

The Z specification of the Production Cell has abstracted away from the low-level details to describe the functionality of the system. To abstract away from the low-level details the system was broken into six components. Each of these components was independently specified capturing local state information and the allowable operations. The operations from the independent specifications were promoted to apply in the complete state and to take into consideration the relationships between components. The final section of the specification extends the partial operations to total operations. The extension was achieved by considering the possible error cases for each partial operation and specifying the error cases in an error schema for the operation. The final total operation is defined by joining of the partial operation and the error schema via the schema calculus.

References

- [1] S. M. Brien and J. E. Nicholls. *Z base standard version 1.0*. Technical Monograph PRG-107, Programming Research Group, Oxford University Computing Laboratory, November 1992.
- [2] C. Lewerentz and T. Lindner. *Case Study Production Cell: A Comparative Study in Formal Software Development*. Lecture Notes in Computer Science. Springer-Verlag, 1994. in press.
- [3] T. Lindner. *Task Description*. In [2].
- [4] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, second edition, 1992.